Đề cương môn học Hệ điều hành
MÔN: HỆ ĐIỀU HÀNH MÃ MÔN HỌC: TH338 SỐ ĐVHT: 3 HỌC KÌ: 6 GIẢNG VIÊN: NGUYỄN PHÚ TRƯỜNG, ĐH Cần Thơ

MỤC ĐÍCH YÊU CẦU

- Giúp sinh viên hiểu được vai trò của hệ điều hành và cơ chế hoạt động của hệ điều hành.
- Cách thiết kế hệ điều hành
- Cách ứng dụng các cơ chế trong việc thiết kế các hệ điều hành hiện đại

KIẾN THỨC NỀN CẦN THIẾT

STT	Nội dung kiến thức nền	Mức độ yêu cầu		
		Tiên quyết	Vận dụng khái niệm/ mô hình	Vận dụng kỹ năng/ phương pháp
1	Kiến trúc máy tính	x		

KIẾN THỨC TOÁN CẦN THIẾT

STT	Nội dung kiến thức	Mức độ yêu cầu			
		Hiểu Khái niệm	Vận dụng Công thức/ định lý	Chứng minh Công thức/ định lý	Vận dụng Phương pháp
1	Giải thuật		x		
2	Ngôn ngữ Pascal và C				x

TÓM TẮT NỘI DUNG MÔN HỌC

- Mô tả các điểm chính yếu của hệ điều hành
- Vai trò và năng lực của hệ điều hành trong hệ thống máy tính.
- Những vấn đề phát sinh trong quá trình thiết kế hệ điều hành cũng như những tiếp cận khác nhau được dùng để phân tích và giải quyết những vấn đề đó.
- Xem xét những chiến lược hệ điều hành phổ biến và cách chúng tác động đến những dịch vụ của các hệ điều hành hiện đại.

ĐỀ CƯƠNG CHI TIẾT CÁC CHƯƠNG

1 MỤC ĐÍCH YÊU CẦU1

2 KIẾN THỨC NỀN CẦN THIẾT1

3 KIẾN THỨC TOÁN CẦN THIẾT1

4 TÓM TẮT NỘI DUNG MÔN HỌC 1

5 ĐỀ CƯƠNG CHI TIẾT CÁC CHƯƠNG1

6 TÀI LIỆU THAM KHẢO3

1 MỤC ĐÍCH YÊU CẦU1

2 KIẾN THỨC NỀN CẦN THIẾT1

TÀI LIỆU THAM KHẢO

1. [Jean Bacon & Tim Harris], Operating Systems, Addison-Wesley, 2003.
2. [Nguyễn Hoàng Việt], Bài giảng Hệ Điều Hành, Khoa CNTT-ĐH Cần Thơ, 1998
3. [Silberschatz, Galvin, Gagne], Operating System Concepts, John Wiley & Sons, 2003
4. [Lê Khắc Nhiên Ân, Hoàng Kiếm], Giáo trình Nhập môn hệ điều hành, Đại học Khoa học Tự nhiên, 2003.
5. [Trần Hạnh Nhi, Hoàng Kiếm], Giáo trình hệ điều hành nâng cao, Đại học Khoa học Tự nhiên,1999.

DUYỆT BỘ MÔN	Ngày 23 tháng 04 năm 2004CÁN BỘ BIÊN SOẠNNguyễn Phú Trường

Giới thiệu môn học Hệ điều hành và bảng thuật ngữ
Hệ điều hành là thành phần quan trọng trong bất cứ hệ thống máy tính nào. Do đó, môn học hệ điều hành là cần thiết cho chương trình giảng dạy sinh viên ngành khoa học máy tính. Giáo trình này mong muốn giới thiệu một cách rõ ràng các khái niệm nằm bên dưới hệ điều hành.

Giáo trình này không tập trung vào một hệ điều hành hay phần cứng cụ thể nào. Thay vào đó, giáo trình sẽ thảo luận những khái niệm cơ bản được áp dụng trong từng hệ điều hành khác nhau.

Để dễ dàng đọc và hiểu giáo trình này, người đọc phải nắm các cấu trúc dữ liệu cơ bản, tổ chức của một hệ thống máy tính và ngôn ngữ lập trình cấp cao (C được dùng để minh họa trong giáo trình). Các khái niệm và giải thuật cơ bản được trình bày trong giáo trình dựa trên việc chúng được dùng trong các hệ điều hành thương mại hay trong các hệ điều hành thử nghiệm.

Giáo trình này gồm có 4 phần với 11 chương:

Phần 1: Tổng quan

Chương 1: Tổng quan về hệ điều hành

Chương 2: Cấu trúc hệ điều hành

Phần 2: Quản lý quá trình

Chương 3: Quá trình

Chương 4: Định thời biểu CPU

Chương 5: Đồng bộ hóa quá trình

Chương 6: Deadlock

Phần 3: Quản lý lưu trữ

Phần 4: Quản lý xuất nhập

Chương	Từ	Ý nghĩa
1	Hand on computer system	Hệ thống máy tính thực hành
	Clustered system	Hệ thống nhóm
	Short-term memory	bộ nhớ lưu ngắn hạn
	Micro-kernel	Vi nhân
2	Spooling	Vùng chứa
	Minidisks	Đĩa nhỏ
3	general-purpose registers	Thanh ghi đa năng

3,5	Long-term scheduler	Bộ định thời dài
	Short-term scheduler	bộ định thời ngắn
	I/O-bound process	Quá trình hướng nhập.xuất
	CPU-bound process	Quá trình hướng xử lý
	bounded capacity	Khả năng chứa bị giới hạn
	unbounded capacity	Khả năng chứa không bị giới hạn
5	CPU burst	Chu kỳ CPU
	Process-control block	Khối điều khiển quá trình
	Nonpreemptive	Không trưng dụng
	Preemptive scheduling	Định thời biểu trưng dụng
	Non-preemptive scheduling	Định thời biểu không trưng dụng
	Dispatcher	Bộ phân phát
	Turn-around time	thời gian hoàn thành
	First-come, First served scheduling	Định thời biểu đến trước, phục vụ trước
	shortest-job-first	Định thời biểu công việc

	scheduling	ngắn nhất trước
	shortest-remaining-time-first	Định thời thời gian còn lại ngắn nhất trước
	priority-scheduling algorithm	giải thuật định thời theo độ ưu tiên
	Starvation	đói CPU
	indefinite blocking	Nghẽn không hạn định
	Aging	Hoá già
	round-robin scheduling algorithm	giải thuật định thời luân phiên
	Time quantum	định mức thời gian
	Processor sharing	chia sẻ bộ xử lý
	multilevel queue-scheduling algorithm	giải thuật định thời hàng đợi nhiều cấp
	multilevel feedback queue scheduling	định thời hàng đợi phản hồi đa cấp
	priority-inheritance protocol	Giao thức kế thừa đội ưu tiên
	analytic evaluation	đánh giá phân tích
	deterministic modeling	mô hình xác định
	queueing-network analysis	phân tích mạng hàng đợi

6	Critical session	Vùng tương trục
	Busy waiting	chờ đợi bận
	Sleep and wakeup	nghẽn và đánh thức
7	Deadlock	Khoá chết
	Mutual exclusion	loại trừ hỗ tương
	Hold and wait	giữ và chờ cấp thêm tài nguyên
	No preemption	Không đòi lại tài nguyên từ quá trình đang giữ chúng
	Circuit wait	Tồn tại chu trình trong đồ thị cấp phát tài nguyên
	claim edge	cạnh thỉnh cầu
8	primitive bare-machine	Máy trơ nguyên thuỷ
	Linkage editor	Bộ soạn thảo liên kết
	Loader	bộ nạp
	Logical address	địa chỉ luận lý
	Physical address	địa chỉ vật lý
	Logical address space	Không gian địa chỉ luận lý
	Physical address space	Không gian địa chỉ vật lý

	Virtual address	địa chỉ ảo
	memory-management unit	bộ quản lý bộ nhớ
	Base register	Thanh ghi nền
	Relocation register	Thanh ghi tái định vị
	Dynamic loading	
	Overlays	Cơ chế phủ lắp
	Two-pass assembler	Trình dịch hợp ngữ hai lần
	Roll in	cuộn vào
	Roll out	cuộn ra
	dispacher	bộ phân phát
	transient operating system code	mã hệ điều hành tạm thời
	Dynamic storage allocation problem	vấn đề cấp phát lưu trữ động
	translation look-aside buffer	
	Wired down	
	Trap	
	Valid bit	Bit hợp lệ

	Invalid bit	Bit không hợp lệ
	forward-mapped page table	bảng trang được ánh xạ chuyển tiếp
	Reentrant code	Mã tái sử dụng
9	Lazy swapper	bộ hoán vị lười
	demand paging	phân trang theo yêu cầu
	Pager	bộ phân trang
	pure demand paging	thuần phân trang theo yêu cầu
	page-fault trap	Trap lỗi trang
	second-chance page-replacement algorithm	Giải thuật thay thế trang cơ hội thứ hai
	(the least frequently used page-replacement algorithm)	Giải thuật thay thế trang được dùng ít thường xuyên nhất
	Working set model	Mô hình tập làm việc
10	user file directory	Thư mục tập tin người dùng
	master file directory	Thư mục tập tin chính
	Acyclic graph	Đồ thị không chứa chu trình
	Symbolic link	liên kết biểu tượng

	Garbage collection	thu dọn rác
	Mounted	Gán vào
	access-control list	danh sách kiểm soát truy xuất
	On-line storage	lưu trữ trực tuyến
	Metadata	Siêu dữ liệu
	Superbock	Siêu khối
	Via	Lưu trữ
	system-wide open-file table	Bảng tập tin đang mở của hệ thống
	Virtual File System	hệ thống tập tin ảo
	Cylinder	Hình trụ
	Track	rãnh
	extent	đoạn mở rộng
	Cluster	nhóm
11	device driver	trình điều khiển thiết bị
	command-ready bit	bit sẵn sàng nhận lệnh
	interrup-request line	dòng yêu cầu ngắt
	interrupt-handler	bộ quản lý ngắt

	nonmaskable interrupt	ngắt không thể che giấu
	maskable interrupt	ngắt có thể giấu
	interrupt vector	vector ngắt
	Interrupt chaining	Vòng ngắt
	interrupt-driven I/O cycle	Chu kỳ nhập/xuất hướng ngắt
	direct memory-access-DMA	bộ điều khiển truy xuất bộ nhớ trực tiếp
	bus-mastering I/O boards	bảng nhập/xuất bus chính
	kernel I/O subsytem	Hệ thống con nhập/xuất của nhân
	interrupt-driven I/O	nhập/xuất hướng ngắt

Cấu trúc hệ điều hành
1 Mục đích Sau khi học xong chương này, người học nắm được những kiến thức sau: - Hiểu hệ điều hành từ ba khía cạnh: người dùng, người lập trình và người thiết kế - Hiểu các dịch vụ mà hệ điều hành cung cấp - Biết các phương pháp được dùng để thiết kế kiến trúc hệ điều hành

Giới thiệu

Hệ điều hành cung cấp môi trường cho các chương trình thực thi. Nội tại, các hệ điều hành rất khác biệt nhau về kiến trúc, chúng được tổ chức cùng với các dòng khác nhau. Thiết kế một hệ điều hành mới là một công việc quan trọng. Các mục đích của hệ thống phải được định nghĩa rõ ràng trước khi thiết kế bắt đầu. Kiểu hệ thống mong muốn là cơ sở cho việc chọn lựa giữa các giải thuật và chiến lược khác nhau.

Hệ điều hành có thể được nhìn từ nhiều lợi điểm khác nhau. Người này xem xét các dịch vụ mà hệ điều hành cung cấp. Người kia quan tâm đến giao diện mà hệ điều hành mang lại cho người dùng và người lập trình. Người khác lại phân rã hệ thống thành những thành phần và các mối quan hệ bên trong của chúng. Trong chương này chúng ta tìm hiểu cả ba khía cạnh của hệ điều hành, thể hiện ba quan điểm của người dùng, người lập trình và người thiết kế hệ điều hành. Chúng ta xem xét các dịch vụ mà hệ điều hành cung cấp, cách chúng được cung cấp và các phương pháp khác nhau được dùng cho việc thiết kế hệ điều hành.

Các thành phần hệ thống

Chúng ta có thể tạo ra một hệ thống lớn và phức tạp như hệ điều hành chỉ khi phân chia hệ điều hành thành những phần nhỏ hơn. Mỗi phần nên là một thành phần được mô tả rõ ràng của hệ thống, với xuất, nhập và các chức năng được định nghĩa cẩn thận. Tuy nhiên, nhiều hệ thống hiện đại chia sẻ mục tiêu hỗ trợ các thành phần hệ thống được liệt kê sau đây:

Quản lý quá trình

Một chương trình không làm gì trừ khi các chỉ thị của nó được thực thi bởi một CPU. Một quá trình có thể được xem như một chương đang thực thi, nhưng định nghĩa của nó sẽ mở rộng khi chúng ta khám phá chi tiết hơn. Một chương trình người dùng được chia thời chẳng hạn như một trình biên dịch là một quá trình. Một chương trình xử lý văn bản đang được thực thi bởi một người dùng trên một PC cũng là một quá trình. Một tác vụ hệ thống, như gởi dữ liệu xuất ra máy in cũng được xem là một quá trình. Bây giờ chúng ta có thể xem xét một quá trình là một công việc hay chương trình chia thời, nhưng chúng ta sẽ nghiên cứu khái niệm này tổng quát hơn trong các chương sau.

Một quá trình cần các tài nguyên xác định-gồm thời gian CPU, bộ nhớ, tập tin, các thiết bị xuất/nhập-để hoàn thành tác vụ của nó. Các tài nguyên này được cấp cho quá trình khi nó được tạo ra, hay được cấp phát tới nó khi nó đang chạy. Ngoài ra, các tài nguyên vật lý và luận lý khác nhau mà quá trình nhận được khi nó được tạo, dữ liệu khởi tạo khác nhau (hay nhập) có thể được truyền qua. Thí dụ, xem xét một quá trình có chức năng hiển thị trạng thái của một tập tin trên màn hình của một thiết bị đầu cuối. Quá trình này sẽ được cho dữ liệu vào là tên của tập tin, và sẽ thực thi các chỉ thị thích hợp và các lời gọi hệ thống đạt được và xuất trên thiết bị cuối thông tin mong muốn. Khi quá trình này kết thúc, hệ điều hành sẽ đòi lại bất cứ tài nguyên nào có thể dùng lại.

Chúng ta nhấn mạnh một chương trình chính nó không phải là một quá trình; một chương trình là một thực thể thụ động, như là nội dung của tập tin được lưu trên đĩa, trái lại một quá trình là một thực thể hoạt động, với một bộ đếm chương trình xác định chỉ thị kế tiếp để thực thi. Việc thực thi của quá trình phải là tuần tự. CPU thực thi một chỉ thị của quá trình sau khi đã thực thi một chỉ thực trước đó cho đến khi quá trình hoàn thành. Ngoài ra, tại bất kỳ thời điểm nào, tối đa một chỉ thị được thực thi cho quá trình. Do đó, mặc dù hai quá trình có thể được liên kết với cùng một quá trình, vì thế chúng được xem như hai chuỗi thực thi riêng. Thông thường có một chương trình sinh ra nhiều quá trình khi nó thực thi.

Một quá trình là một đơn vị công việc trong hệ thống. Một hệ thống chứa tập các quá trình, một vài quá trình này là các quá trình hệ điều hành (thực thi mã hệ thống) và các quá trình còn lại là các quá trình người dùng (chúng thực thi mã người dùng). Tất cả các quá trình này có tiềm năng thực thi đồng hành bằng cách đa hợp CPU giữa các quá trình.

Hệ điều hành có nhiệm vụ cho các hoạt động sau khi đề cập đến chức năng quản lý quá trình:

- Tạo và xoá các quá trình người dùng và hệ thống
- Tạm dừng và thực thi tiếp quá trình
- Cung cấp các cơ chế đồng bộ hoá quá trình
- Cung cấp các cơ chế giao tiếp quá trình
- Cung cấp cơ chế quản lý deadlock

Quản lý bộ nhớ chính

Bộ nhớ chính là trung tâm điều hành của một máy tính hiện đại. Bộ nhớ chính là một mảng các từ (words) hay bytes có kích thước lớn từ hàng trăm ngàn tới hàng tỉ. Mỗi từ hay byte có địa chỉ riêng. Bộ nhớ chính là một kho chứa dữ liệu có khả năng truy xuất nhanh được chia sẻ bởi CPU và các thiết bị xuất/nhập. Bộ xử lý trung tâm đọc các chỉ thị từ bộ nhớ trong chu kỳ lấy chỉ thị, nó đọc và viết dữ liệu từ bộ nhớ chính trong chu kỳ lấy dữ liệu. Bộ nhớ chính thường là thiết bị lưu trữ lớn mà CPU có thể định địa chỉ và truy xuất trực tiếp. Thí dụ, đối với CPU xử lý dữ liệu từ đĩa, dữ liệu trước tiên được chuyển tới bộ nhớ chính bởi lời gọi xuất/nhập được sinh ra bởi CPU. Tương tự, các chỉ thị phải ở trong bộ nhớ cho CPU thực thi chúng.

Đối với một chương trình được thực thi, nó phải được ánh xạ các địa chỉ và được nạp vào bộ nhớ. Khi chương trình thực thi, nó truy xuất các chỉ thị chương trình và dữ liệu từ bộ nhớ bằng cách tạo ra các địa chỉ tuyệt đối này. Cuối cùng, chương trình kết thúc, không gian bộ nhớ của nó được khai báo sẵn, và chương trình có thể được nạp và thực thi.

Để cải tiến việc sử dụng CPU và tốc độ đáp ứng của máy tính cho người dùng, chúng ta phải giữ nhiều chương trình vào bộ nhớ. Nhiều cơ chế quản lý bộ nhớ khác nhau được dùng và tính hiệu quả của các giải thuật phụ thuộc vào từng trường hợp cụ thể. Chọn một cơ chế quản lý bộ nhớ cho một hệ thống xác định phụ thuộc vào nhiều yếu tố-đặc biệt trên thiết kế phần cứng của hệ thống. Mỗi giải thuật đòi hỏi sự hỗ trợ phần cứng của nó.

Hệ điều hành có nhiệm vụ cho các hoạt động sau khi đề cập tới việc quản lý bộ nhớ

- Giữ vết về phần nào của bộ nhớ hiện đang được dùng và quá trình nào đang dùng.
- Quyết định quá trình nào được nạp vào bộ nhớ khi không gian bộ nhớ trở nên sẵn dùng.
- Cấp phát và thu hồi không gian bộ nhớ khi được yêu cầu.

Quản lý tập tin

Quản lý tập tin là một trong những thành phần có thể nhìn thấy nhất của hệ điều hành. Máy tính có thể lưu thông tin trên nhiều loại phương tiện lưu trữ vật lý khác nhau. Băng từ, đĩa từ, đĩa quang là những phương tiện thông dụng nhất. Mỗi phương tiện này có đặc điểm và tổ chức riêng. Mỗi phương tiện được điều khiển bởi một thiết bị, như một ổ đĩa hay ổ băng từ. Các thuộc tính này bao gồm tốc độ truy xuất, dung lượng, tốc độ truyền dữ liệu và phương pháp truy xuất (tuần tự hay ngẫu nhiên).

Nhờ vào việc sử dụng thuận lợi hệ thống máy tính, hệ điều hành cung cấp tầm nhìn luận lý của việc lưu trữ thông tin đồng nhất. Hệ điều hành trừu tượng hoá các thuộc tính vật lý của các thiết bị lưu trữ để định nghĩa một đơn vị lưu trữ luận lý là tập tin. Hệ điều hành ánh xạ các tập tin trên các thiết bị lưu trữ vật lý, và truy xuất các tập tin này bằng các thiết bị lưu trữ.

Tập tin là tập hợp thông tin có quan hệ được định nghĩa bởi người tạo. Thông thường, các tập tin biểu diễn chương trình và dữ liệu. Các tập tin dữ liệu có thể là số, chữ cái, chữ số. Các tập tin có dạng bất kỳ (thí dụ, các tập tin văn bản) hay có thể được định dạng có cấu trúc (thí dụ, các trường cố định). Một tập tin chứa một chuỗi các bits, bytes, các dòng hay các mẫu tin mà ý nghĩa của nó được định nghĩa bởi người tạo. Khái niệm tập tin là một khái niệm cực kỳ thông dụng.

Hệ điều hành cài đặt một khái niệm trừu tượng của tập tin bằng cách quản lý phương tiện lưu trữ như đĩa, băng từ và các thiết bị điều khiển chúng. Các tập tin cũng thường được tổ chức trong các thư mục để dễ dàng sử dụng chúng. Cuối cùng, khi nhiều người dùng truy xuất tập tin, chúng ta muốn kiểm soát ai và trong cách gì (thí dụ: đọc, viết, chèn,..) các tập tin có thể được truy xuất.

Hệ điều hành có nhiệm vụ thực hiện các hoạt động trong việc quản lý hệ thống tập tin:

- Tạo và xoá tập tin
- Tạo và xoá thư mục
- Hỗ trợ các hàm nguyên thuỷ để thao tác tập tin và thư mục
- Ánh xạ các tập tin trên các thiết bị lưu trữ phụ
- Sao lưu dự phòng tập tin trên các phương tiện lưu trữ ổ định

Quản lý hệ thống xuất/nhập

Một trong những mục đích của hệ điều hành là che giấu sự khác biệt của các thiết bị phần cứng từ người dùng. Thí dụ, trong UNIX sự khác biệt của các thiết bị xuất/nhập bị che giấu từ phần chính của hệ điều hành bởi các hệ thống con xuất/nhập. Hệ thống con xuất/nhập chứa:

- Thành phần quản lý bộ nhớ chứa vùng đệm (buffering), lưu trữ (caching) và spooling (vùng chứa).
- Giao diện trình điều khiển thiết bị chung.
- Trình điều khiển cho các thiết bị xác định.

Chỉ trình điều khiển thiết bị biết sự khác biệt của các thiết bị xác định mà nó được gán

Quản lý việc lưu trữ phụ

Mục đích chính của một hệ thống máy tính là thực thi các chương trình. Những chương trình này với dữ liệu chúng truy xuất phải nằm trong bộ nhớ chính hay lưu trữ chính trong quá trình thực thi. Vì bộ nhớ chính quá nhỏ để lưu tất cả dữ liệu và chương trình và vì dữ liệu quản lý bị mất khi mất điện, hệ thống máy tính phải cung cấp việc lưu trữ phụ để lưu dự phòng bộ nhớ chính. Hầu hết các hệ thống máy tính hiện đại dùng đĩa như phương tiện lưu trữ trực tuyến cho cả chương trình và dữ liệu. Hầu hết các chương trình – gồm trình biên dịch, trình dịch hợp ngữ, thủ tục sắp xếp, trình soạn thảo và trình định dạng – được lưu trên đĩa cho tới khi được nạp vào trong bộ nhớ và sau đó dùng đĩa khi cả hai nguồn và đích của việc xử lý. Do đó, quản lý hợp lý việc lưu trữ đĩa có vai trò quan trọng đối với một hệ thống máy tính.

Hệ điều hành có nhiệm vụ thực hiện các hoạt động sau trong việc quản lý đĩa:

- Quản lý không gian trống
- Cấp phát lưu trữ
- Định thời đĩa

Vì lưu trữ phụ được dùng thường xuyên nên nó phải được dùng một cách hiệu quả. Tốc độ toàn bộ của các thao tác của máy tính có thể xoay quanh tốc độ hệ thống con đĩa và các giải thuật thao tác trên hệ thống con đó.

Mạng

Hệ phân tán là tập hợp các bộ xử lý, chúng không chia sẻ bộ nhớ, các thiết bị ngoại vi hay đồng hồ. Thay vào đó mỗi bộ xử lý có bộ nhớ, đồng hồ và các bộ xử lý giao tiếp với nhau thông qua các đường giao tiếp như bus tốc độ cao hay mạng. Các bộ xử lý trong hệ thống phân tán khác nhau về kích thước và chức năng. Chúng có thể chứa các bộ vi xử lý, trạm làm việc, máy vi tính và các hệ thống máy tính thông thường.

Các bộ xử lý trong hệ thống được nối với nhau thông qua mạng truyền thông có thể được cấu hình trong nhiều cách khác nhau. Mạng có thể được nối kết một phần hay toàn bộ. Thiết kế mạng truyền thông phải xem xét vạch đường thông điệp và các chiến lược nối kết, và các vấn đề cạnh tranh hay bảo mật.

Hệ thống phân tán tập hợp những hệ thống vật lý riêng rẻ, có thể có kiến trúc không đồng nhất thành một hệ thống chặt chẻ, cung cấp người dùng với truy xuất tới các tài nguyên khác nhau mà hệ thống duy trì. Truy xuất tới các tài nguyên chia sẻ cho phép tăng tốc độ tính toán, chức năng, khả năng sẵn dùng của dữ liệu, khả năng tin cậy. Hệ điều hành thường tổng quát hoá việc truy xuất mạng như một dạng truy xuất tập tin, với những chi tiết mạng được chứa trong trình điều khiển thiết bị của giao diện mạng. Các giao thức tạo một hệ thống phân tán có thể có một ảnh hưởng to lớn trên tiện ích và tính phổ biến của hệ thống đó. Sự đổi mới của World Wide Web đã tạo ra một phương pháp truy xuất mới cho thông tin chia sẻ. Nó đã cải tiến giao thức truyền tập tin (File Transfer Protocol-FTP) và hệ thống tập tin mạng (Network File System-NFS) đã có bằng cách xoá yêu cầu cho một người dùng đăng nhập trước khi người dùng đó được phép dùng tài nguyên ở xa. Định nghĩa một giao thức mới, giao thức truyền siêu văn bản (hypertext transfer protocol-http), dùng trong giao tiếp giữa một trình phục vụ web và trình duyệt web. Trình duyệt web chỉ cần gởi yêu cầu thông tin tới một trình phục vụ web của máy ở xa, thông tin (văn bản, đồ hoạ, liên kết tới những thông tin khác) được trả về.

Hệ thống bảo vệ

Nếu một hệ thống máy tính có nhiều người dùng và cho phép thực thi đồng hành của nhiều quá trình, thì các quá trình khác nhau phải được bảo vệ từ các hoạt động của quá trình khác. Cho mục đích này, các cơ chế đảm bảo rằng các tập tin, phân đoạn bộ nhớ, CPU, và các tài nguyên khác có thể được điều hành chỉ bởi các quá trình có quyền phù hợp từ hệ điều hành.

Thí dụ, phần cứng định địa chỉ bộ nhớ đảm bảo rằng một quá trình có thể thực thi chỉ trong không gian địa chỉ của chính nó. Bộ định thời đảm bảo rằng không có quá trình nào có thể đạt được điều khiển của CPU mà cuối cùng không trả lại điều khiển. Các thanh ghi điều khiển thiết bị không thể truy xuất tới người dùng vì thế tính đúng đắn của các thiết bị ngoại vi khác nhau được bảo vệ.

Bảo vệ là một cơ chế để điều khiển truy xuất của các chương trình, quá trình hay người dùng tới tài nguyên được định nghĩa bởi một hệ thống máy tính. Cơ chế này phải cung cấp phương tiện để đặc tả các điều khiển được áp đặt và phương tiện cho việc ép buộc.

Bảo vệ có thể cải tiến khả năng tin cậy bằng cách phát hiện các lỗi tiềm tàng tại các giao diện giữa các hệ thống con thành phần. Phát hiện các lỗi giao diện sớm thường có thể ngăn chặn nguy cơ ảnh hưởng tới hệ thống con bởi một hệ thống con khác. Tài nguyên không được bảo vệ không thể ngăn chặn việc sử dụng bởi người dùng không có quyền. Hệ thống hướng bảo vệ (protection-oriented system) cung cấp một phương tiện để phân biệt giữa việc dùng có quyền và không có quyền.

Hệ thống thông dịch lệnh

Một trong những chương trình hệ thống quan trọng nhất đối với hệ điều hành là trình thông dịch lệnh. Nó là giao diện giữa người dùng và hệ điều hành. Một vài hệ điều hành chứa trình thông dịch lệnh trong nhân (kernel). Các hệ điều hành khác nhau như MS-DOS và UNIX xem trình thông dịch lệnh như một chương trình đặc biệt đang chạy khi một công

việc được khởi tạo hay khi người dùng đăng nhập lần đầu tiên (trên các hệ thống chia thời).

Nhiều lệnh (commands) được cung cấp tới hệ điều hành bởi các lệnh điều khiển (control statements). Khi một công việc mới được bắt đầu trong hệ thống bó, hay khi một người dùng đăng nhập tới hệ thống chia thời, một chương trình đọc và thông dịch các câu lệnh điều khiển được thực thi tự động. Chương trình này còn được gọi trình thông dịch thẻ điều khiển (control-card interpreter) hay trình thông dịch dòng lệnh và thường được biết như shell. Chức năng của nó đơn giản là: lấy câu lệnh tiếp theo và thực thi nó.

Các hệ điều hành thường khác nhau trong vùng shell, với một trình thông dịch lệnh thân thiện với người dùng làm cho hệ thống có thể chấp nhập nhiều hơn đối với người dùng. Một dạng giao diện thân thiện người dùng là hệ thống trình đơn-cửa sổ trên cơ sở chuột (mouse-based window-and-menu system) được dùng trong Macintosh và Microsoft Windows. Chuột được di chuyển tới vị trí con trỏ chuột trên ảnh hay biểu tượng trên màn hình biểu diễn các chương trình, tập tin, và các hàm hệ thống. Phụ thuộc vào vị trí con trỏ chuột, nhấn một nút trên chuột có thể nạp một chương trình, chọn một tập tin hay thư mục hay kéo xuống một trình đơn chứa các câu lệnh. Các shell mạnh hơn, phức tạp hơn và khó học hơn được đánh giá cao bởi một số người dùng khác. Trong những shell này, các lệnh được đánh vào từ bàn phím được hiển thị trên màn hình hay in ra thiết bị đầu cuối, với phím enter (hay return) chỉ rằng một lệnh hoàn thành và sẵn sàng được thực thi. Shell của MS-DOS và UNIX điều hành theo cách này.

Các câu lệnh giải quyết việc tạo và quản lý quá trình, quản lý xuất/nhập, quản lý việc lưu trữ phụ, quản lý bộ nhớ chính, truy xuất hệ thống tập tin, bảo vệ và mạng.

Các dịch vụ hệ điều hành

Hệ điều hành cung cấp một môi trường cho việc thực thi các chương trình. Nó cung cấp các dịch vụ xác định tới chương trình và tới người

dùng của các chương trình đó. Dĩ nhiên, các dịch vụ được cung cấp khác nhau từ hệ điều hành này với hệ điều hành kia nhưng chúng có thể xác định các lớp chung. Các dịch vụ hệ điều hành được cung cấp sự tiện dụng cho người lập trình để thực hiện tác vụ lập trình dễ dàng.

- Thực thi chương trình: hệ thống phải có thể nạp chương trình vào bộ nhớ và chạy chương trình đó. Chương trình phải có thể kết thúc việc thực thi của nó bình thường hay không bình thường (hiển thị lỗi).
- Thao tác xuất/nhập: một chương trình đang chạy có thể yêu cầu xuất/nhập. Xuất/nhập này có thể liên quan tới tập tin hay thiết bị xuất/nhập. Đối với các thiết bị cụ thể, các chức năng đặc biệt có thể được mong muốn (như quay lại từ đầu một ổ băng từ, hay xoá màn hình). Đối với tính hiệu quả và tính bảo vệ, người dùng thường không thể điều khiển các thiết bị xuất/nhập trực tiếp. Do đó, hệ điều hành phải cung cấp một phương tiện để thực hiện xuất/nhập..
- Thao tác hệ thống tập tin: hệ thống tập tin có sự quan tâm đặc biệt. Các chương trình cần đọc từ và viết tới các tập tin. Chương trình cũng cần tạo và xoá tập tin bằng tên.
- Giao tiếp: trong nhiều trường hợp, một quá trình cần trao đổi thông tin với các quá trình khác. Giao tiếp như thế có thể xảy ra trong hai cách chính. Cách đầu tiên xảy ra giữa các quá trình được thực thi trên cùng máy tính; cách thứ hai xảy ra giữa hai quá trình đang được thực thi trên các máy tính khác nhau được kết nối với nhau bởi một mạng máy tính. Các giao tiếp có thể được thực hiện bằng bộ nhớ được chia sẻ, hay bằng kỹ thuật truyền thông điệp, trong đó các gói tin được di chuyển giữa các quá trình bởi hệ điều hành.
- Phát hiện lỗi: hệ điều hành liên tục yêu cầu nhận biết các lỗi có thể phát sinh. Các lỗi có thể xảy ra trong CPU và phần cứng bộ nhớ (như lỗi bộ nhớ hay lỗi về điện), trong các thiết bị xuất/nhập (như lỗi chẩn lẻ trên băng từ, lỗi nối kết mạng, hết giấy in) và trong chương trình người dùng (như tràn số học, cố gắng truy xuất một vị trí bộ nhớ không hợp lệ, dùng quá nhiều thời gian CPU). Đối với mỗi loại lỗi, hệ điều hành nên thực hiện một hoạt động hợp lý để đảm bảo tính toán đúng và không đổi.

Ngoài ra, một tập chức năng khác của hệ điều hành tồn tại không giúp người dùng, nhưng đảm bảo các điều hành hữu hiệu của chính hệ thống. Các hệ thống với nhiều người dùng có thể đạt tính hữu hiệu bằng cách chia sẻ tài nguyên máy tính giữa các người dùng.

- Cấp phát tài nguyên: khi nhiều người dùng đăng nhập vào hệ thống hay nhiều công việc đang chạy cùng lúc, tài nguyên phải được cấp tới mỗi người dùng. Nhiều loại tài nguyên khác nhau được quản lý bởi hệ điều hành. Một số tài nguyên (như chu kỳ CPU, bộ nhớ chính, lưu trữ tập tin) có mã cấp phát đặt biệt, trái lại các tài nguyên khác (như thiết bị xuất/nhập) có mã yêu cầu và giải phóng thường hơn. Thí dụ, xác định cách tốt nhất để dùng CPU, hệ điều hành có các thủ tục định thời biểu CPU. Các thủ tục này xem xét tốc độ CPU, các công việc phải được thực thi, số thanh ghi sẵn dùng và các yếu tố khác. Cũng có các thủ tục cấp phát ổ băng từ để dùng cho một công việc. Một thủ tục như thế định vị ổ băng từ chưa được dùng và đánh dấu một bảng bên trong để ghi người dùng mới của ổ băng từ. Một thủ tục khác được dùng để xoá bảng đó. Các thủ tục này cũng có thể cấp phát các máy vẽ, modem, các thiết bị ngoại vi khác.
- Tính toán: chúng ta muốn giữ vết người dùng nào sử dụng bao nhiêu và loại tài nguyên máy tính nào. Giữ vết này có thể được dùng để tính toán (tính tiền người dùng) hay đơn giản thống kê sử dụng. Thống kê sử dụng có thể là công cụ có giá trị cho người nghiên cứu muốn cấu hình lại hệ thống để cải tiến các dịch vụ tính toán.
- Bảo vệ: người sở hữu thông tin được lưu trong hệ thống máy tính đa người dùng muốn điều khiển thông tin này. Khi nhiều quá trình riêng rẻ thực thi đồng hành, không thể cho một quá trình can thiệp tới các quá trình khác hay tới chính hệ điều hành. Bảo vệ đảm bảo rằng tất cả truy xuất tài nguyên của hệ thống được kiểm soát. An toàn hệ thống từ người dùng bên ngoài cũng là vấn đề quan trọng. An toàn bắt đầu với mỗi người dùng có quyền đối với hệ thống, thường bằng mật khẩu để được phép truy xuất tài nguyên. Mở rộng việc bảo vệ đối với các thiết bị xuất/nhập bên ngoài, bao gồm modem, card mạng từ những truy xuất không hợp lệ, và ghi lại các

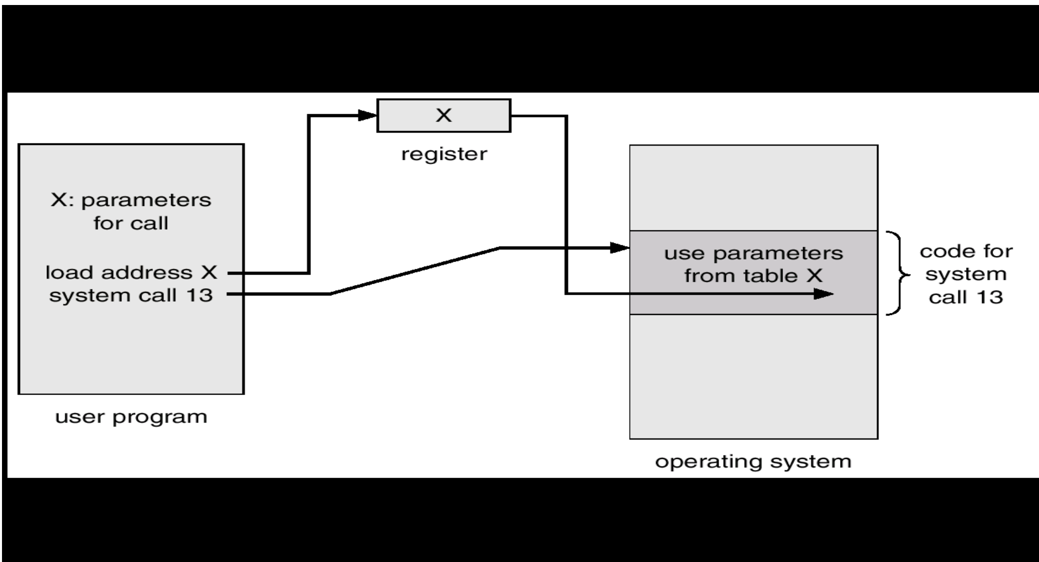
nối kết để phát hiện đột nhập vào hệ thống. Nếu hệ thống bảo vệ và bảo mật, những cảnh báo phải được thiết lập xuyên suốt.

Lời gọi hệ thống

Lời gọi hệ thống cung cấp giao diện giữa một quá trình và hệ điều hành. Các lời gọi này thường sẵn dùng như các chỉ thị hợp ngữ và chúng thường được liệt kê trong những tài liệu hướng dẫn sử dụng được dùng bởi những người lập trình hợp ngữ.

Những hệ thống xác định cho phép lời gọi hệ thống được thực hiện trực tiếp từ một chương trình ngôn ngữ cấp cao, trong đó các lời gọi thường tương tự lời gọi hàm hay thủ tục được định nghĩa trước. Chúng có thể tạo ra một lời gọi tới một chương trình con tại thời điểm thực thi cụ thể.

Lời gọi hệ thống xảy ra trong nhiều cách khác nhau, phụ thuộc vào máy tính đang dùng. Thường nhiều thông tin được yêu cầu hơn là đơn giản xác định lời gọi hệ thống mong muốn. Loại chính xác và lượng thông tin khác nhau dựa vào hệ điều hành và lời gọi cụ thể. Thí dụ, để nhập dữ liệu, chúng ta có thể cần xác định tập tin hay thiết bị dùng như nguồn nhập, địa chỉ và chiều dài vùng đệm bộ nhớ mà dữ liệu nhập sẽ được đọc vào. Dĩ nhiên, thiết bị hay tập tin và chiều dài có thể ẩn trong lời gọi.



Hình II-1-Truyền tham số như bảng

Có ba phương pháp thông dụng để truyền tham số tới hệ điều hành. Phương pháp đơn giản nhất là truyền tham số trong các thanh ghi. Trong một vài trường hợp, các tham số thường lưu trữ trong một khối hay bảng trong bộ nhớ và địa chỉ của khối được truyền như một tham số trong thanh ghi (Hình II.1). Các tham số cũng có thể được thay thế, hay được đẩy vào trong ngăn xếp bởi chương trình, và được lấy ra khỏi ngăn xếp bởi hệ điều hành. Một vài hệ điều hành dùng phương pháp khối hay ngăn xếp vì các phương pháp này không giới hạn số lượng hay chiều dài của tham số đang được truyền.

Các chương trình hệ thống

Một khía cạnh khác của hệ thống hiện đại là tập hợp các chương trình hệ thống. Xem lại hình I.1, nó minh họa cấu trúc phân cấp máy tính luận lý. Tại cấp thấp nhất là phần cứng. Kế đến là hệ điều hành, sau đó các chương trình hệ thống và cuối cùng là các chương trình ứng dụng. Các chương trình hệ thống cung cấp môi trường thuận lợi cho việc phát triển và thực thi chương trình. Vài trong chúng là các giao diện người dùng đơn giản cho các lời gọi hệ thống; các hệ thống còn lại được xem xét phức tạp hơn. Chúng có thể được chia thành các loại sau:

- Quản lý tập tin: các chương trình tạo, xóa, chép, đổi tên, in, kết xuất, liệt kê, và các thao tác tập tin thư mục thông thường.
- Thông tin trạng thái: một vài chương trình đơn giản yêu cầu hệ thống ngày, giờ, lượng bộ nhớ hay đĩa sẵn dùng, số lượng người dùng, hay thông tin trạng thái tương tự. Sau đó, thông tin được định dạng và được in tới thiết bị đầu cuối hay thiết bị xuất khác hoặc tập tin.
- Thay đổi tập tin: nhiều trình soạn thảo văn bản có thể sẵn dùng để tạo và thay đổi nội dung của tập tin được lưu trên đĩa hay băng từ.
- Hỗ trợ ngôn ngữ lập trình: trình biên dịch, trình hợp ngữ và trình thông dịch cho các ngôn ngữ lập trình thông dụng (như C, C++, Java, Visual Basic và PERL) thường được cung cấp tới người dùng với hệ điều hành. Hiện nay, một vài chương trình này được cung cấp riêng và có giá cả riêng.
- Nạp và thực thi chương trình: một khi chương trình được tập hợp hay được biên dịch, nó phải được nạp vào bộ nhớ để được thực thi. Hệ thống có thể cung cấp bộ nạp tuyệt đối, bộ nạp có thể tái định vị, bộ soạn thảo liên kết và bộ nạp phủ lắp. Các hệ thống gỡ rối cho các ngôn ngữ cấp cao hay ngôn ngữ máy cũng được yêu cầu.
- Giao tiếp: các chương trình này cung cấp cơ chế tạo các nối kết ảo giữa các quá trình, người dùng, các hệ thống máy tính khác. Chúng cho phép người dùng gởi các thông điệp tới màn hình của người dùng khác, hiển thị các trang web, gởi thư điện tử, đăng nhập từ xa hay để chuyển các tập tin từ máy tính này tới máy tính khác.

Nhiều hệ điều hành được cung cấp với các chương trình giải quyết các vấn đề giao tiếp thông thường hay thực hiện các thao tác phổ biến. Những chương trình như thế gồm các trình duyệt Web, bộ xử lý văn bản và bộ định dạng văn bản, hệ cơ sở dữ liệu, trình biên dịch, các gói phần mềm đồ họa và phân tích thống kê, trò chơi,..Những chương trình này được gọi là các tiện ích hệ thống hay chương trình ứng dụng.

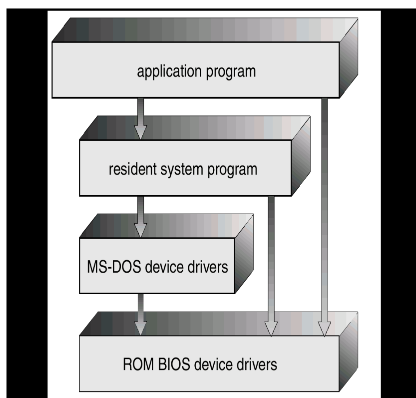
Hầu hết người dùng nhìn hệ điều hành như các chương trình hệ thống hơn các lời gọi hệ thống thực sự. Nghĩ về việc sử dụng một PC. Khi máy tính của chúng ta chạy hệ điều hành Microsoft Windows, chúng ta có thể thấy một trình thông dịch dòng lệnh MS-DOS hay giao diện cửa sổ

và trình đơn đồ họa. Cả hai sử dụng cùng một tập lời gọi hệ thống như lời gọi hệ thống trông rất khác và hoạt động trong các cách khác nhau. Do đó, tầm nhìn của chúng ta về thực chất có thể bị tách rời với cấu trúc hệ thống thực sự. Vì thế, thiết kế một giao diện tiện dụng và thân thiện không là một chức năng trực tiếp của hệ điều hành. Trong giáo trình này chúng ta sẽ tập trung các vấn đề cơ bản của việc cung cấp dịch vụ đầy đủ cho các chương trình người dùng. Từ quan điểm hệ điều hành, chúng ta không phân biệt sự khác nhau giữa các chương trình người dùng và các chương trình hệ thống.

Cấu trúc hệ thống

Một hệ thống lớn và phức tạp như một hệ điều hành hiện đại phải được xây dựng cẩn thận nếu nó thực hiện chức năng hợp lý và được hiệu chỉnh dễ dàng. Một phương pháp thông thường là chia tác vụ thành các thành phần nhỏ hơn là có một hệ thống nguyên khối. Mỗi modules này nên là một thành phần hoàn toàn xác định với nhập, xuất, chức năng được định nghĩa cẩn thận. Trong phần này chúng ta sẽ thảo luận về cách thức mà các thành phần được nối kết và trộn lẫn trong một nhân.

Cấu trúc đơn giản

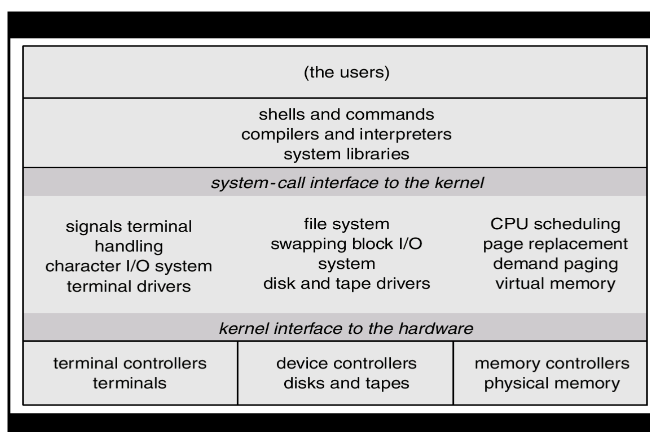


Hình II-2-Cấu trúc phân tầng của MS-DOS

Nhiều hệ thống thương mại không có kiến trúc rõ ràng. Thường các hệ điều hành như thế được bắt đầu như các hệ thống nhỏ, đơn giản và có giới hạn. Sau đó chúng lớn lên ngoài giới hạn mã ban đầu của chúng.

MS-DOS là một thí dụ cho hệ thống dạng này. Ban đầu, nó được thiết kế và thực hiện bởi một vài người mà họ không tưởng rằng chúng sẽ trở nên quá phổ biến. Nó được viết để cung cấp các khả năng nhiều nhất trong không gian ít nhất (vì bị giới hạn bởi phần cứng mà nó đang chạy) vì nó không được phân chia thành các modules một cách cẩn thận. Hình II.2, hiển thị cấu trúc của nó.

UNIX là một hệ điều hành khác mà ban đầu nó bị giới hạn bởi chức năng phần cứng. Nó chứa hai phần có thể tách rời nhau: nhân và các chương trình hệ thống. Nhân lại được chia thành một loạt các giao diện và trình điều khiển thiết bị mà chúng được thêm vào và mở rộng qua nhiều năm khi UNIX được cải tiến. Chúng ta có thể hiển thị hệ điều hành UNIX truyền thống khi nó được phân tầng như hình II.3. Mọi thứ bên dưới giao diện lời gọi hệ thống và bên trên phần cứng vật lý là nhân. Nhân cung cấp hệ thống tập tin, bộ định thời CPU, quản lý bộ nhớ và các chức năng khác của hệ điều hành thông qua lời gọi hệ thống. Có rất nhiều chức năng được nối kết trong cấp thứ nhất. Điều này làm cho UNIX khó có thể nâng cấp khi những thay đổi trong phần một ảnh hưởng bất lợi cho những phần khác.



Hình II-3 – Cấu trúc hệ thống của UNIX

Lời gọi hệ thống định nghĩa giao diện lập trình ứng dụng (API-Application Programming Interface) cho UNIX; tập hợp các chương trình hệ thống thường sẵn dùng định nghĩa giao diện người dùng. Người lập trình và giao diện người dùng định nghĩa ngữ cảnh mà nhân phải hỗ trợ.

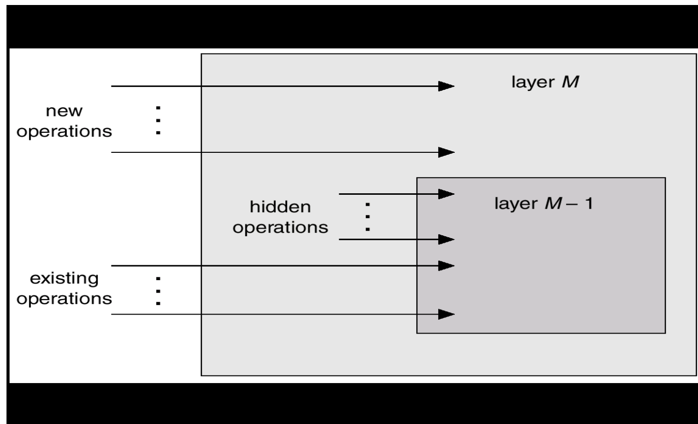
Những ấn bản mới của UNIX được thiết kế để dùng phần cứng tiên tiến hơn. Được cung cấp sự hỗ trợ phần cứng hợp lý, các hệ điều hành có thể được chia thành nhiều phần nhỏ hơn và phù hợp hơn là các hệ thống MS-DOS và UNIX ban đầu. Sau đó, các hệ điều hành có thể giữ lại việc điều khiển lớn hơn qua máy tính và qua các ứng dụng thực hiện việc sử dụng máy tính đó. Những người cài đặt thoải mái hơn trong việc thực hiện những thay đổi các hoạt động bên trong của hệ thống và trong việc tạo các hệ điều hành theo module. Dưới phương pháp từ trên-xuống (top-down), toàn bộ các chức năng và đặc điểm được xác định và được chia thành các thành phần. Sự phân chia này cho phép các người lập trình che giấu thông tin; do đó họ tự do cài đặt các thủ tục cấp thấp khi họ thấy thích hợp, được cung cấp các giao diện bên ngoài của các thủ tục không bị thay đổi do chính thủ tục đó thực hiện các tác vụ thông thường.

Phương pháp phân tầng

Việc phân chia từng phần của một hệ thống có thể được thực hiện trong nhiều cách. Một trong những phương pháp này là thực hiện tiếp cận phân tầng. Trong tiếp cận này hệ điều hành được chia thành nhiều tầng (hay cấp), mỗi tầng được xây dựng trên đỉnh của tầng dưới nó. Tầng cuối cùng (tầng 0) là phần cứng; tầng cao nhất (tầng N) là giao diện người dùng.

Một tầng hệ điều hành là sự cài đặt của một đối tượng trừu tượng. Đối tượng trừu tượng này là sự bao gói dữ liệu và các điều hành có thể thao tác dữ liệu đó. Một tầng hệ điều hành điển hình –tầng M- được mô tả trong hình II.4. Nó chứa các cấu trúc dữ liệu và tập hợp các thủ tục có

thể được gọi bởi các tầng cấp cao hơn. Sau đó, tầng M có thể gọi các thao tác trên tầng cấp thấp hơn.



Hình II-4-Một tầng hệ điều hành

Lợi điểm chủ yếu của tiếp cận phân tầng là tính module. Các tầng được chọn dựa trên cơ sở tầng trên sử dụng chức năng (hay các điều hành) và các dịch vụ chỉ của tầng cấp dưới nó. Tiếp cận này đơn giản hóa việc gỡ rối và kiểm tra hệ thống. Tầng đầu tiên có thể được gỡ rối mà không có bất cứ sự quan tâm nào cho phần còn lại của hệ thống. Bởi vì theo định nghĩa, nó chỉ sử dụng phần cứng cơ bản để cài đặt các chức năng của nó. Một khi tầng đầu tiên được gỡ rối, chức năng sửa lỗi của nó có thể được đảm đương trong khi tầng thứ hai được gỡ rối, …Nếu một lỗi được tìm thấy trong khi gỡ rối cho một tầng xác định, lỗi phải được nằm trên tầng đó vì các tầng bên dưới đã được gỡ rối rồi. Do đó, thiết kế và cài đặt hệ thống được đơn giản hóa khi hệ thống được phân chia thành nhiều tầng.

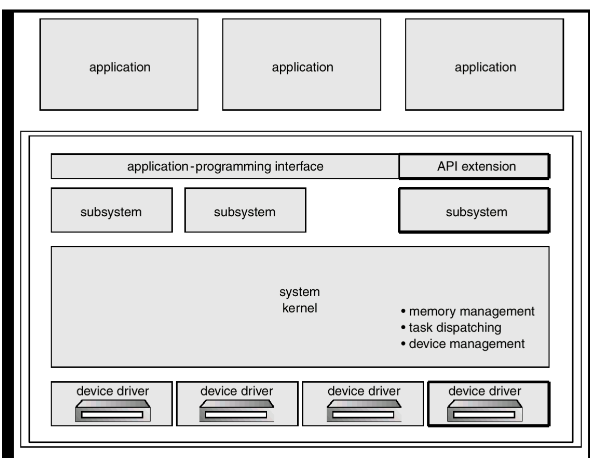
Mỗi tầng được cài đặt chỉ với các thao tác được cung cấp bởi các tầng bên dưới. Một tầng không cần biết các thao tác được cài đặt như thế nào; nó chỉ cần biết các thao tác đó làm gì. Do đó, mỗi tầng che giấu sự tồn tại của cấu trúc dữ liệu, thao tác và phần cứng từ các tầng cấp cao hơn.

Khó khăn chính của tiếp cận phân tầng liên quan tới việc định nghĩa cẩn thận các tầng vì một tầng chỉ có thể sử dụng các tầng bên dưới nó. Thí

dụ, trình điều khiển thiết bị cho không gian đĩa được dùng bởi các giải thuật bộ nhớ ảo phải nằm ở tại cấp thấp hơn trình điều khiển thiết bị của các thủ tục quản lý bộ nhớ vì quản lý bộ nhớ yêu cầu khả năng sử dụng không gian đĩa.

Các yêu cầu có thể không thật sự rõ ràng. Thường thì các trình điều khiển lưu trữ dự phòng nằm trên bộ định thời CPU vì trình điều khiển cần phải chờ nhập/xuất và CPU có thể được định thời lại trong thời gian này. Tuy nhiên, trên hệ thống lớn, bộ định thời có thể có nhiều thông tin hơn về tất cả quá trình đang hoạt động hơn là có thể đặt vừa trong bộ nhớ. Do đó, thông tin này có thể cần được hoán vị vào và ra bộ nhớ, yêu cầu thủ tục trình điều khiển lưu trữ dự phòng nằm bên dưới bộ định thời CPU.

Vấn đề cuối cùng với các cài đặt phân tầng là chúng có khuynh hướng ít hiệu quả hơn các loại khác. Thí dụ, khi chương trình người dùng thực thi thao tác nhập/xuất, nó thực thi một lời gọi hệ thống. Lời gọi hệ thống này được bẫy (trapped) tới tầng nhập/xuất, nó yêu cầu tầng quản lý bộ nhớ, sau đó gọi tầng định thời CPU, sau đó được truyền tới phần cứng. Tại mỗi tầng, các tham số có thể được hiệu chỉnh, dữ liệu có thể được truyền,…Mỗi tầng thêm chi phí cho lời gọi hệ thống; kết quả thực sự là lời gọi hệ thống mất thời gian lâu hơn khi chúng thực hiện trên hệ thống không phân tầng.



Hình II-5 Cấu trúc phân tầng của OS/2

Những giới hạn này gây một phản ứng nhỏ chống lại việc phân tầng trong những năm gần đây. Rất ít các tầng với nhiều chức năng được thiết kế, cung cấp nhiều lợi điểm của mã được module trong khi tránh những vấn đề khó khăn của định nghĩa và giao tiếp tầng. Thí dụ, OS/2 bổ sung thêm tính năng đa tác vụ và điều hành hai chế độ cùng một số đặc điểm mới. Vì tính phức tạp được bổ sung và phần cứng mạnh hơn mà OS/2 được thiết kế, hệ thống được cài đặt trong dạng phân tầng.

Vi nhân (Microkernels)

Khi hệ điều hành UNIX được mở rộng, nhân trở nên lớn và khó quản lý. Vào giữa những năm 1980, các nhà nghiên cứu tại đại học Carnegie Mellon phát triển một hệ điều hành được gọi là Match mà module hóa nhân dùng tiếp cận vi nhân (micro kernel). Phương pháp này định kiến trúc của hệ điều hành bằng xóa tất cả thành phần không quan trọng từ nhân và cài chúng như các chương trình cấp người dùng và hệ thống. Kết quả này làm cho nhân nhỏ hơn. Có rất ít sự nhất trí liên quan đến việc quyết định dịch vụ nào nên để lại trong nhân và dịch vụ nào nên được cài đặt trong không gian người dùng. Tuy nhiên, thường thì các vi nhân điển hình cung cấp quá trình và quản lý bộ nhớ tối thiểu ngoài phương tiện giao tiếp.

Chức năng chính của vi nhân là cung cấp tiện nghi giao tiếp giữa chương trình khách hàng và các dịch vụ khác mà chúng đang chạy trong không gian người dùng. Giao tiếp được cung cấp bằng truyền thông điệp. Thí dụ, nếu chương trình khách hàng muốn truy xuất một tập tin, nó phải giao tiếp với trình phục vụ tập tin (file server). Chương trình người dùng và dịch vụ không bao giờ giao tiếp trực tiếp. Đúng hơn là chúng giao tiếp gián tiếp bằng cách truyền thông điệp với vi nhân.

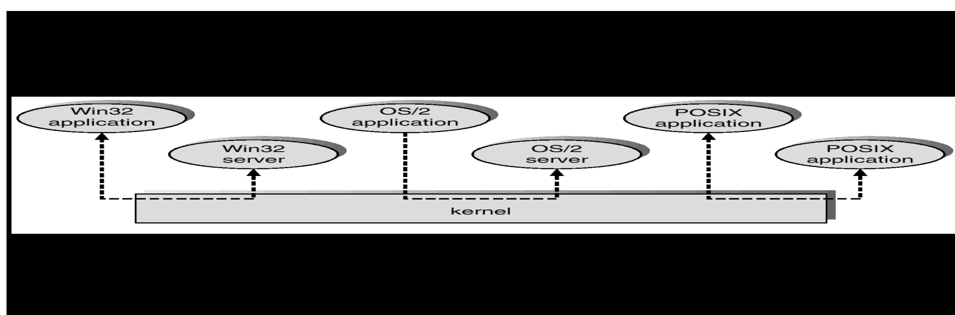
Thuận lợi của tiếp cận vi nhân là dễ dàng mở rộng hệ điều hành. Tất cả dịch vụ mới được thêm tới không gian người dùng và do đó không yêu cầu phải hiệu chỉnh nhân. Kết quả là hệ điều hành dễ dàng hơn để

chuyển đổi từ thiết kế phần cứng này sang thiết kế phần cứng khác. Vi nhân cũng cung cấp khả năng an toàn và tin cậy hơn vì hầu hết các dịch vụ đang chạy như người dùng –hơn là nhân- các quá trình. Nếu một dịch vụ bị lỗi, phần còn lại của hệ điều hành vẫn không bị ảnh hưởng.

Một số hệ điều hành hiện đại dùng tiếp cận vi nhân. Tru64 UNIX (Digital UNIX trước đây) cung cấp giao diện UNIX tới người dùng, nhưng nó được cài đặt với nhân Mach. Nhân Mach ánh xạ các lời gọi hệ thống vào các thông điệp tới các dịch vụ cấp người dùng tương ứng. Hệ điều hành Apple MacOS Server được dựa trên cơ sở nhân Mach.

QNX là hệ điều hành thời thực cũng dựa trên cơ sở thiết kế vi nhân. Vi nhân QNX cung cấp các dịch vụ cho việc truyền thông điệp và định thời quá trình. Nó cũng quản lý giao tiếp mạng cấp thấp và các ngắt phần cứng. Tất cả dịch vụ khác trong QNX được cung cấp bởi các quá trình chuẩn chạy bên ngoài nhân trong chế độ người dùng.

Windows NT dùng một cấu trúc tổng hợp. Windows NT được thiết kế để chạy các ứng dụng khác nhau, gồm Win32 (ứng dụng thuần Windows), OS/2, và POSIX (Portable Operating System Interface for uniX). Nó cung cấp một server chạy trong không gian người dùng cho mỗi loại ứng dụng. Các chương trình khách hàng cho mỗi loại ứng dụng chạy trong không gian người dùng. Nhân điều phối việc truyền thông điệp giữa các ứng dụng khách hàng và server ứng dụng. Cấu trúc client-server của Windows NT được mô tả trong hình II.6



Hình II-6 – Cấu trúc client-server của Windows NT

Máy ảo

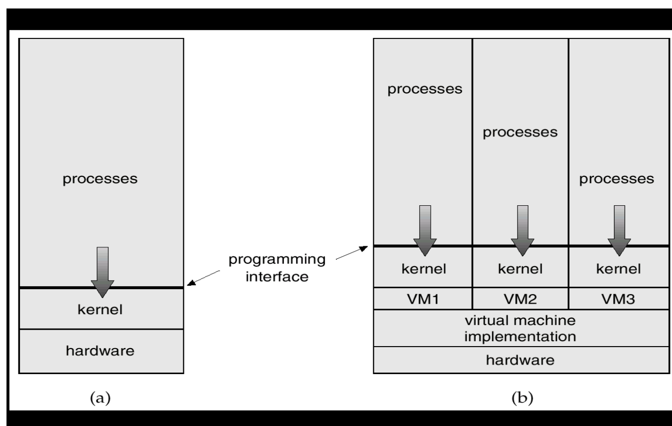
Về mặt khái niệm, một hệ thống máy tính được cấu thành từ các tầng. Phần cứng là cấp thấp nhất trong tất cả hệ thống như thế. Nhân chạy tại cấp kế tiếp dùng các chỉ thị phần cứng để tạo một tập lời gọi hệ thống cho việc sử dụng các tầng bên ngoài. Do đó, các chương trình hệ thống trên nhân có thể dùng các lời gọi hệ thống hay các chỉ thị phần cứng. Trong nhiều trường hợp, các chương trình này không có sự khác biệt giữa hai cách thực hiện. Do đó, mặc dù chúng được truy xuất khác nhau, nhưng cả hai cung cấp chức năng mà chương trình có thể dùng để tạo thậm chí nhiều chức năng tiên tiến hơn. Sau đó, các chương trình hệ thống xem phần cứng và các lời gọi hệ thống như chúng đang ở cùng một cấp.

Một vài hệ thống thực hiện cơ chế này một cách chi tiết hơn bằng cách cho phép các chương trình hệ thống được gọi dễ dàng bởi các chương trình ứng dụng. Trước đó, mặc dù các chương trình hệ thống ở tại cấp cao hơn các thủ tục khác, nhưng các chương trình ứng dụng có thể hiển thị mọi thứ dưới chúng trong cấu trúc phân cấp như là một phần của chính máy đó. Tiếp cận phân tầng này được đưa đến một kết luận luận lý trong khái niệm máy ảo (virtual machine). Một hệ điều hành máy ảo cho các hệ thống IBM là một thí dụ điển hình nhất về khái niệm máy ảo vì IBM tiên phong thực hiện trong lĩnh vực này.

Bằng cách sử dụng bộ định thời CPU và kỹ thuật bộ nhớ ảo, một hệ điều hành có thể tạo một hình ảnh mà một quá trình có bộ xử lý của chính nó với bộ nhớ (ảo) của chính nó. Dĩ nhiên, thường thì một quá trình có các đặc điểm khác nhau, như các lời gọi hệ thống và hệ thống tập tin, mà không được cung cấp bởi phần cứng trơ. Thêm vào đó, tiếp cận máy ảo không cung cấp bất kỳ chức năng bổ sung nào; nhưng đúng hơn là cung cấp một giao diện giống hệt như phần cứng trơ ở bên dưới. Mỗi quá trình được cung cấp với một bản sao (ảo) của máy tính bên dưới (Hình II.7).

Một khó khăn chính với tiếp cận máy ảo liên quan đến hệ thống đĩa. Giả sử rằng máy vật lý có ba ổ đĩa nhưng muốn hỗ trợ bảy máy ảo. Rõ ràng,

nó không thể cấp phát một ổ đĩa tới mỗi máy ảo. Nhớ rằng chính phần mềm máy ảo sẽ cần không gian đĩa liên tục để cung cấp bộ nhớ ảo. Giải pháp này cung cấp đĩa ảo, mà nó đúng trong tất cả khía cạnh ngoại trừ kích thước-được thuật ngữ hóa đĩa nhỏ (minidisks) trong hệ điều hành máy ảo của IBM. Hệ thống cài đặt nhiều đĩa nhỏ bằng cách cấp phát nhiều rảnh ghi trên đĩa vật lý như là các đĩa nhỏ khi cần. Hiển nhiên, tổng kích thước của tất cả đĩa nhỏ là nhỏ hơn kích thước của không gian đĩa vật lý sẵn có.



Hình II-7-Các mô hình hệ thống. (a) Máy không ảo. (b) máy ảo

Do đó, người dùng được cho máy ảo của chính họ. Sau đó, họ có thể chạy bất kỳ hệ điều hành hay gói phần mềm nào sẵn dùng trên phần cứng bên dưới. Đối với hệ thống IBM VM, một người dùng thường chạy CMS-một hệ điều hành giao tiếp đơn người dùng. Phần mềm máy ảo được quan tâm với đa máy ảo đa chương trên một máy vật lý nhưng không cần xem xét bất cứ phần mềm hỗ trợ người dùng. Việc sắp xếp này có thể cung cấp một sự phân chia hữu ích thành hai phần nhỏ hơn của vấn đề thiết kế một hệ thống giao tiếp đa người dùng.

Cài đặt

Mặc dù khái niệm máy ảo là hữu ích nhưng rất khó cài đặt. Nhiều công việc được yêu cầu cung cấp một bản sao chính xác của máy bên dưới. Máy bên dưới có hai chế độ: chế độ người dùng và chế độ kiểm soát. Phần mềm máy ảo có thể chạy trong chế độ kiểm soát vì nó là hệ điều hành. Chính máy ảo có thể thực thi chỉ trong chế độ người dùng. Tuy nhiên, chỉ khi máy vật lý có hai chế độ thì nó mới là máy ảo. Do đó, chúng ta phải có một chế độ người dùng ảo và một chế độ kiểm soát ảo. Cả hai đều chạy trong chế độ người dùng vật lý. Các hoạt động đó gây ra sự chuyển từ chế độ người dùng tới chế độ kiểm soát trên một máy thật (như lời gọi hệ thống hay cố gắng thực thi một chỉ thị được cấp quyền) cũng phải gây ra sự chuyển đổi từ chế độ người dùng ảo tới chế độ kiểm soát ảo trên một máy ảo.

Lợi điểm

Có hai lợi điểm chính trong việc sử dụng máy ảo. Thứ nhất, bằng cách bảo vệ hoàn toàn các tài nguyên hệ thống, máy ảo cung cấp mức độ bảo mật cao. Thứ hai, máy ảo cho phép phát triển hệ thống được thực hiện mà không cần phá vỡ hoạt động hệ thống thông thường.

Mỗi máy ảo hoàn toàn bị cô lập từ các máy ảo khác, vì thế chúng ta không gặp phải bất kỳ vấn đề bảo mật nào như tài nguyên hệ thống khác hoàn toàn được bảo vệ. Thí dụ, các ứng dụng không được tin cậy được tải về từ Internet có thể được chạy trong một máy ảo riêng. Một bất lợi của môi trường này là không có sự chia sẻ tài nguyên trực tiếp. Hai tiếp cận cung cấp sự chia sẻ được cài đặt. Thứ nhất, có thể chia sẻ một đĩa nhỏ. Cơ chế này được làm mẫu sau một đĩa được chia sẻ vật lý. Thứ hai, có thể định nghĩa một mạng của các máy ảo, mỗi máy ảo có thể gửi thông tin qua các mạng giao tiếp này nhưng nó được cài đặt bằng phần mềm.

Những hệ thống máy ảo như thế là một phương tiện truyền thông hữu hiệu cho việc nghiên cứu và phát triển hệ điều hành. Thông thường, thay đổi một hệ điều hành là một tác vụ khó. Vì các hệ điều hành là các chương trình lớn và phức tạp, sự thay đổi trên một phần này có thể gây

một lỗi khó hiểu trong những phần khác. Sức mạnh của hệ điều hành làm cho trường hợp này là cực kỳ nguy hiểm. Vì hệ điều hành thực thi trong chế độ kiểm soát, một thay đổi sai trong một con trỏ có thể gây lỗi và có thể phá hủy toàn hệ thống tập tin. Do đó, cần phải kiểm tra tất cả thay đổi của hệ điều hành một cách cẩn thận.

Tuy nhiên, hệ điều hành chạy trên máy và điều khiển hoàn toàn máy đó. Do đó, hệ thống hiện hành phải bị dừng và ngừng việc sử dụng trong khi những thay đổi được thực hiện và kiểm tra. Thời điểm này thường được gọi là thời gian phát triển hệ thống. Vì nó làm cho hệ thống không sẵn dùng đối với người sử dụng nên thời gian phát triển hệ thống thường được lập thời biểu vào buổi tối hay cuối tuần, khi tải hệ thống thấp.

Một hệ thống máy ảo có thể loại trừ nhiều vấn đề này. Người lập trình hệ thống được cung cấp chính máy ảo của họ, và phát triển hệ thống được thực hiện trên máy ảo thay vì trên máy vật lý thật sự. Một hệ điều hành thông thường ít khi bị phá vỡ vì phát triển hệ thống. Mặc dù những thuận lợi này, nhưng rất ít cải tiến trên kỹ thuật này được thực hiện gần đây.

Tóm tắt

Hệ điều hành cung cấp một số dịch vụ. Tại cấp thấp nhất, lời gọi hệ thống cho phép một chương trình đang chạy thực hiện yêu cầu trực tiếp từ hệ thống. Tại cấp cao hơn, trình thông dịch cung cấp cơ chế cho người dùng đưa ra yêu cầu mà không viết chương trình. Các lệnh có thể xuất phát từ tập tin trong suốt thời gian thực thi theo chế độ xử lý theo lô, hay trực tiếp từ bàn phím trong chế độ tương tác hay chia sẻ thời gian. Các chương trình hệ thống được cung cấp để thoả mãn nhiều yêu cầu thông thường của người dùng.

Các loại yêu cầu khác nhau dựa theo cấp yêu cầu. Cấp gọi hệ thống phải cung cấp các chức năng cơ bản, như điều khiển quá trình, quản lý tập tin và thiết bị. Các yêu cầu cấp cao hơn được thoả mãn bởi trình thông dịch lệnh và chương trình hệ thống được dịch vào một chuỗi các lời gọi hệ thống. Các dịch vụ hệ thống có thể được phân cấp thành nhiều loại: điều

khiển chương trình, yêu cầu trạng thái, yêu cầu nhập/xuất. Lỗi chương trình được xem xét các yêu cầu ẩn cho dịch vụ.

Một khi dịch vụ hệ thống được định nghĩa, cấu trúc của hệ điều hành được phát triển. Các bảng khác nhau cần được ghi thông tin định nghĩa trạng thái của hệ thống máy tính và trạng thái của công việc hệ thống.

Thiết kế một hệ điều hành mới là công việc rất quan trọng. Thiết kế hệ thống như thứ tự của các tầng hay sử dụng vi nhân được xem là một kỹ thuật tốt. Khái niệm máy ảo thực hiện tiếp cận phân tầng và xem nhân của hệ điều hành và phần cứng như là phần cứng của nó. Các hệ điều hành khác có thể được nạp trên đỉnh của máy ảo.

Quá trình
Mục đích Sau khi học xong chương này, người học nắm được những kiến thức sau: - Hiểu các khái niệm về quá trình - Hiểu cách lập thời biểu quá trình - Biết các thao tác trên quá trình Hiểu cách giao tiếp liên quá trình

Giới thiệu

Những hệ thống máy tính ban đầu cho phép chỉ một chương trình được thực thi tại một thời điểm. Chương trình này có toàn quyền điều khiển hệ thống và có truy xuất tới tất cả tài nguyên của hệ thống. Những hệ thống máy tính hiện nay cho phép nhiều chương trình được nạp vào bộ nhớ và được thực thi đồng hành. Sự phát triển này yêu cầu sự điều khiển mạnh mẽ hơn và phân chia nhiều hơn giữa các quá trình. Yêu cầu này dẫn đến khái niệm quá trình, một chương trình đang thực thi. Quá trình là một đơn vị công việc trong một hệ điều hành chia thời hiện đại.

Một hệ điều hành phức tạp hơn được mong đợi nhiều hơn trong việc thực hiện các hành vi của người dùng. Mặc dù quan tâm chủ yếu của hệ điều hành là thực thi chương trình người dùng, nhưng nó cũng quan tâm đến các tác vụ khác nhau bên ngoài nhân. Do đó, một hệ thống chứa tập hợp các quá trình: quá trình hệ điều hành thực thi mã hệ thống, quá trình người dùng thực thi mã người dùng. Tất cả quá trình này có tiềm năng thực thi đồng hành, với một CPU (hay nhiều CPU) được đa hợp giữa chúng. Bằng cách chuyển đổi CPU giữa các quá trình, hệ điều hành có thể làm cho máy tính hoạt động với năng suất cao hơn.

Khái niệm quá trình

Một vấn đề cần thảo luận là cái gì được gọi trong tất cả hoạt động của CPU? Một hệ thống bó thực thi công việc, trái lại một hệ thống chia thời thực thi chương trình người dùng hay tác vụ. Thậm chí trên hệ thống đơn người dùng như Microsoft Windows và Macintosh OS, một người dùng có thể chạy nhiều chương trình tại một thời điểm: bộ xử lý văn bản, trình duyệt web, e-mail. Thậm chí nếu người dùng có thể thực thi chỉ một quá

trình tại một thời điểm, thì một hệ điều hành cần hỗ trợ những hoạt động được lập trình bên trong, như quản lý bộ nhớ. Trong nhiều khía cạnh, tất cả hoạt động là tương tự vì thế chúng ta gọi tất cả chúng là quá trình.

Quá trình

Thật vậy, một quá trình là một chương trình đang thực thi. Một quá trình không chỉ là mà chương trình, nó còn bao gồm hoạt động hiện hành như được hiện diện bởi giá trị của bộ đếm chương trình và nội dung các thanh ghi của bộ xử lý. Ngoài ra, một quá trình thường chứa ngăn xếp quá trình, chứa dữ liệu tạm thời (như các tham số phương thức, các địa chỉ trả về, các biến cục bộ) và phần dữ liệu chứa các biến toàn cục.

Chúng ta nhấn mạnh rằng, một chương trình không phải là một quá trình; một chương trình là một thực thể thụ động, như nội dung của các tập tin được lưu trên đĩa, trái lại một quá trình là một thực thể chủ động, với một bộ đếm chương trình xác định chỉ thị lệnh tiếp theo sẽ thực thi và tập hợp tài nguyên có liên quan.

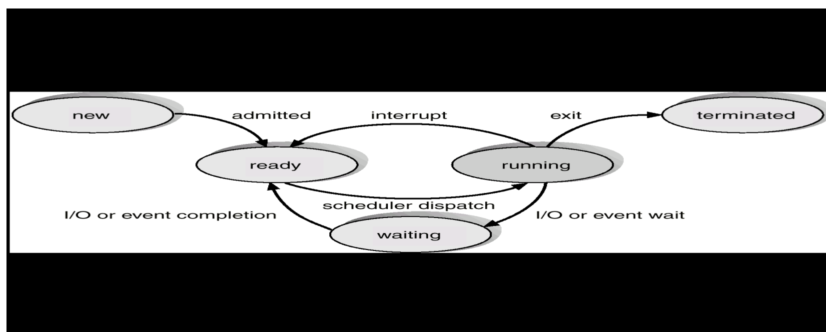
Mặc dù hai quá trình có thể được liên kết với cùng chương trình nhưng chúng được chứa hai thứ tự thực thi riêng rẻ. Thí dụ, nhiều người dùng có thể đang chạy các bản sao của chương trình gởi nhận thư, hay cùng người dùng có thể nạp lên nhiều bản sao của một chương trình soạn thảo văn bản. Mỗi bản sao của chúng là một quá trình riêng và mặc dù các phần văn bản là giống nhau, các phần dữ liệu khác nhau. Ngoài ra, một quá trình có thể tạo ra nhiều quá trình khi nó thực thi.

Trạng thái quá trình

Khi một quá trình thực thi, nó thay đổi trạng thái. Trạng thái của quá trình được định nghĩa bởi các hoạt động hiện hành của quá trình đó. Mỗi quá trình có thể ở một trong những trạng thái sau:

- Mới (new): quá trình đang được tạo ra
- Đang chạy (running): các chỉ thị đang được thực thi
- Chờ (waiting): quá trình đang chờ sự kiện xảy ra (như hoàn thành việc nhập/xuất hay nhận tín hiệu)
- Sẳn sàng (ready): quá trình đang chờ được gán tới một bộ xử lý.
- Kết thúc (terminated): quá trình hoàn thành việc thực thi

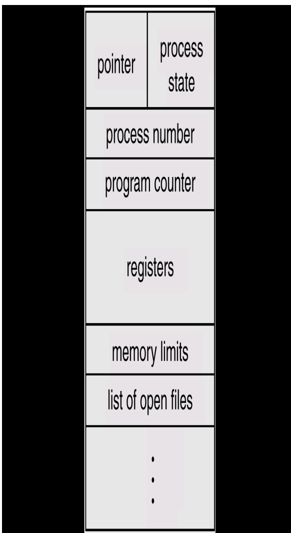
Các tên trạng thái này là bất kỳ, và chúng khác nhau ở các hệ điều hành khác nhau. Tuy nhiên, các trạng thái mà chúng hiện diện được tìm thấy trên tất cả hệ thống. Các hệ điều hành xác định mô tả trạng thái quá trình. Chỉ một quá trình có thể đang chạy tức thì trên bất kỳ bộ xử lý nào mặc dù nhiều quá trình có thể ở trạng thái sẳn sàng và chờ.



Hình III-1-Lưu đồ trạng thái quá trình

Khối điều khiển quá trình

Mỗi quá trình được hiện diện trong hệ điều hành bởi một khối điều khiển quá trình (Process Control Block-PCB) – cũng được gọi khối điều khiển tác vụ. Một PCB được hiển thị trong hình III-2. Nó chứa nhiều phần thông tin được gắn liền với một quá trình xác định, gồm:

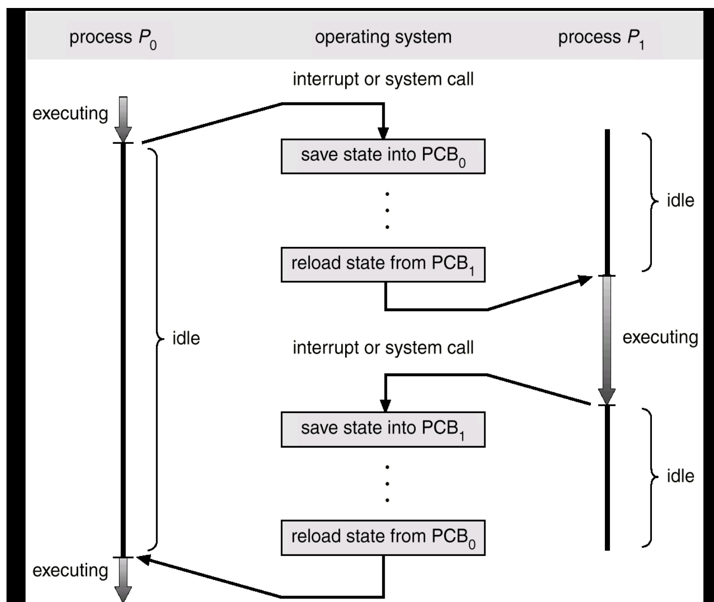


Hình III-2-Khối điều khiển quá trình

- Trạng thái quá trình (process state): trạng thái có thể là mới, sẵn sàng, đang chạy, chờ đợi, kết thúc, …
- Bộ đếm chương trình (program counter): bộ đếm hiển thị địa chỉ của chỉ thị kế tiếp được thực thi cho quá trình này.
- Các thanh ghi (registers) CPU: các thanh ghi khác nhau về số lượng và loại, phụ thuộc vào kiến trúc máy tính. Chúng gồm các bộ tổng (accumulators), các thanh ghi chỉ mục, các con trỏ ngăn xếp, và các thanh ghi đa năng (general-purpose registers), cùng với thông tin mã điều kiện (condition-code information). Cùng với bộ đếm chương trình, thông tin trạng thái này phải được lưu khi một ngắt xảy ra, cho phép quá trình được tiếp tục một cách phù hợp sau đó (Hình III.3).
- Thông tin lập thời biểu CPU (CPU-scheduling information): thông tin gồm độ ưu tiên của quá trình, các con trỏ chỉ tới các hàng đợi lập thời biểu, và bất kỳ tham số lập thời biểu khác.
- Thông tin quản lý bộ nhớ (Memory-management information): thông tin này có thể gồm những thông tin như giá trị của các thanh ghi nền và thanh ghi giới hạn, các bảng trang hay các bảng phân đoạn, phụ thuộc hệ thống bộ nhớ được dùng bởi hệ điều hành.
- Thông tin tính toán (accounting information): thông tin này gồm lượng CPU và thời gian thực được dùng, công việc hay số quá trình,…

- Thông tin trạng thái nhập/xuất (I/O status information): thông tin này gồm danh sách của thiết bị nhập/xuất được cấp phát quá trình này, một danh sách các tập tin đang mở,..

PCB đơn giản phục vụ như kho chứa cho bất cứ thông tin khác nhau từ quá trình này tới quá trình khác.



Hình III-3-Lưu đồ hiển thị việc chuyển CPU từ quá trình này tới quá trình khác

Luồng

Mô hình quá trình vừa được thảo luận ngụ ý rằng một quá trình là một chương trình thực hiện một luồng đơn thực thi. Thí dụ, nếu một quá trình đang chạy một chương trình xử lý văn bản, một luồng đơn của chỉ thị đang được thực thi. Đây là một luồng điều khiển đơn cho phép quá trình thực thi chỉ một tác vụ tại một thời điểm. Thí dụ, người dùng không thể cùng lúc nhập các ký tự và chạy bộ kiểm tra chính tả trong cùng một quá trình. Nhiều hệ điều hành hiện đại mở rộng khái niệm quá trình để

cho phép một quá trình có nhiều luồng thực thi. Do đó, chúng cho phép thực hiện nhiều hơn một tác vụ tại một thời điểm.

Lập thời biểu quá trình

Mục tiêu của việc đa chương là có vài quá trình chạy tại tất cả thời điểm để tận dụng tối đa việc sử dụng CPU. Mục tiêu của chia thời là chuyển CPU giữa các quá trình thường xuyên để người dùng có thể giao tiếp với mỗi chương trình trong khi đang chạy. Một hệ thống đơn xử lý chỉ có thể chạy một quá trình. Nếu nhiều hơn một quá trình tồn tại, các quá trình còn lại phải chờ cho tới khi CPU rảnh và có thể lập thời biểu lại.

Hàng đợi lập thời biểu

Khi các quá trình được đưa vào hệ thống, chúng được đặt vào hàng đợi công việc. Hàng đợi chứa tất cả quá trình trong hệ thống. Các quá trình đang nằm trong bộ nhớ chính sẵn sàng và chờ để thực thi được giữ trên một danh sách được gọi là hàng đợi sẵn sàng. Hàng đợi này thường được lưu như một danh sách liên kết. Đầu của hàng đợi sẵn sàng chứa hai con trỏ: một chỉ đến PCB đầu tiên và một chỉ tới PCB cuối cùng trong danh sách. Chúng ta bổ sung thêm trong mỗi PCB một trường con trỏ chỉ tới PCB kế tiếp trong hàng đợi sẵn sàng.

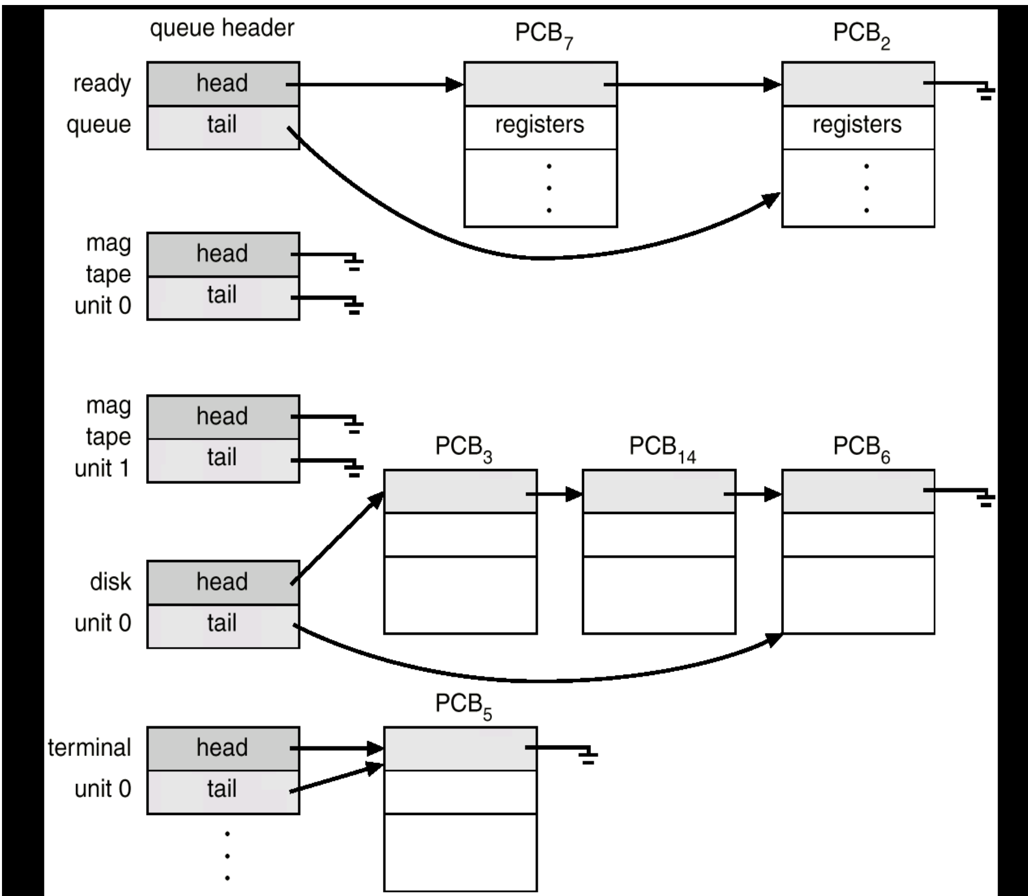
Hệ điều hành cũng có các hàng đợi khác. Khi một quá trình được cấp phát CPU, nó thực thi một khoảng thời gian và cuối cùng kết thúc, được ngắt, hay chờ một sự kiện xác định xảy ra, chẳng hạn như hoàn thành một yêu cầu nhập/xuất. Trong trường hợp yêu cầu nhập/xuất, một yêu cầu có thể là ổ đĩa băng từ tận hiến hay một thiết bị được chia sẻ như đĩa. Vì hệ thống có nhiều quá trình, đĩa có thể bận với yêu cầu nhập/xuất của các quá trình khác. Do đó, quá trình phải chờ đĩa. Danh sách quá trình chờ một thiết bị nhập/xuất cụ thể được gọi là hàng đợi thiết bị. Mỗi thiết bị có hàng đợi của chính nó như hình III.4.

Một biểu diễn chung của lập thời biểu quá trình là một lưu đồ hàng đợi, như hình III.5. Mỗi hình chữ nhật hiện diện một hàng đợi. Hai loại hàng đợi được hiện diện: hàng đợi sẳn sàng và tập các hàng đợi thiết bị, vòng tròn hiện diện tài nguyên phục vụ hàng đợi, các mũi tên hiển thị dòng các quá trình trong hệ thống.

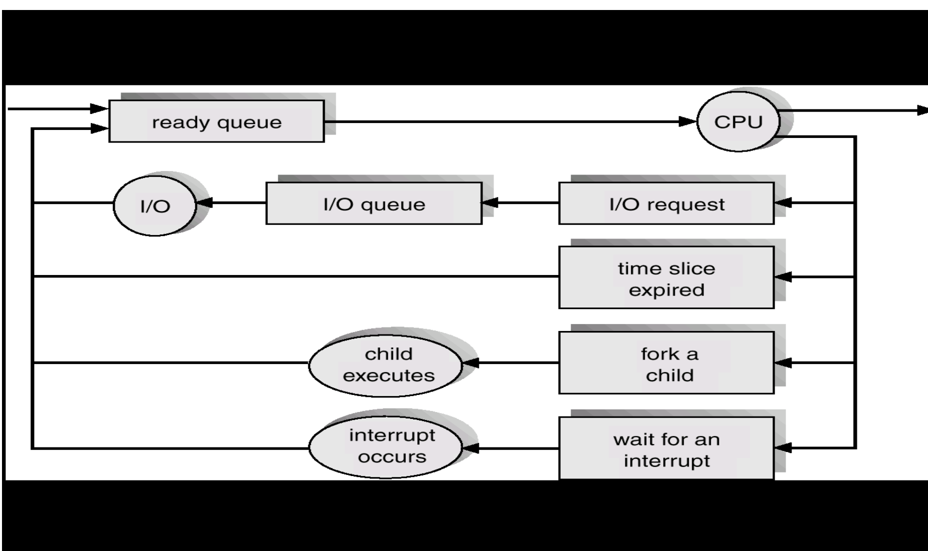
Một quá trình mới khởi đầu được đặt vào hàng đợi. Nó chờ trong hàng đợi sẳn sàng cho tới khi nó được chọn thực thi. Một khi quá trình được gán tới CPU và đang thực thi, một trong nhiều sự kiện có thể xảy ra:

- Quá trình có thể phát ra một yêu cầu nhập/xuất và sau đó được đặt vào trong hàng đợi nhập/xuất.
- Quá trình có thể tạo một quá trình con và chờ cho quá trình con kết thúc
- Khi một ngắt xảy ra, quá trình có thể bị buộc trả lại CPU và được đặt trở lại trong hàng đợi sẳn sàng.

Trong hai trường hợp đầu, cuối cùng quá trình chuyển từ trạng thái chờ tới trạng thái sẳn sàng và sau đó đặt trở lại vào hàng đợi sẳn sàng. Một quá trình tiếp tục chu kỳ này cho tới khi nó kết thúc. Tại thời điểm kết thúc nó bị xoá từ tất cả hàng đợi. Sau đó, PCB và tài nguyên của nó được thu hồi.



Hình III-4-Hàng đợi sẳn sàng và các hàng đợi nhập/xuất khác nhau



Hình III-5-Biểu diễn lưu đồ hàng đợi của lập thời biểu quá trình

Bộ định thời biểu

Một quá trình di dời giữa hai hàng đợi định thời khác nhau suốt thời gian sống của nó. Hệ điều hành phải chọn, cho mục đích định thời, các quá trình từ các hàng đợi này. Quá trình chọn lựa này được thực hiện bởi bộ định thời hợp lý.

Trong hệ thống bó, thường nhiều hơn một quá trình được gởi đến hơn là có thể được thực thi tức thì. Các quá trình này được lưu tới thiết bị lưu trữ (như đĩa), nơi chúng được giữ cho việc thực thi sau đó. Bộ định thời dài (long-term scheduler) hay bộ định thời công việc (job scheduler), chọn các quá trình từ vùng đệm và nạp chúng vào bộ nhớ để thực thi. Bộ định thời ngắn (short-term scheduler) hay bộ định thời CPU chọn một quá trình từ các quá trình sẵn sàng thực thi và cấp phát CPU cho quá trình đó.

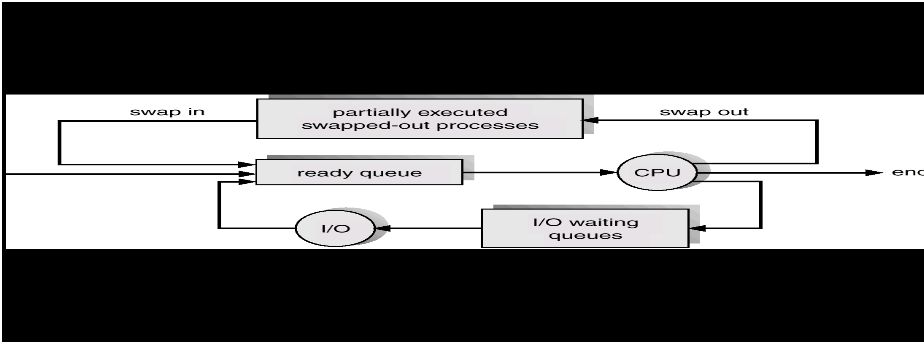
Sự khác biệt chủ yếu giữa hai bộ định thời là tính thường xuyên của việc thực thi. Bộ định thời CPU phải chọn một quá trình mới cho CPU thường xuyên. Một quá trình có thể thực thi chỉ một vài mili giây trước khi chờ yêu cầu nhập/xuất. Bộ định thời CPU thường thực thi ít nhất một lần mỗi 100 mili giây. Vì thời gian ngắn giữa việc thực thi nên bộ định thời phải nhanh. Nếu nó mất 10 mili giây để quyết định thực thi một quá trình 100 mili giây thì 10/(100+10) = 9 phần trăm của CPU đang được dùng (hay bị lãng phí) đơn giản cho định thời công việc.

Ngược lại, bộ định thời công việc thực thi ít thường xuyên hơn. Có vài phút giữa việc tạo các quá trình mới trong hệ thống. Bộ định thời công việc điều khiển mức độ đa chương – số quá trình trong bộ nhớ. Nếu mức độ đa chương ổn định thì tốc độ trung bình của việc tạo quá trình phải bằng tốc độ khởi hành trung bình của quá trình rời hệ thống. Vì khoảng thời gian dài hơn giữa việc thực thi nên bộ định thời công việc có thể cấp nhiều thời gian hơn để chọn một quá trình thực thi.

Bộ định thời công việc phải thực hiện một chọn lựa cẩn thận. Thông thường, hầu hết các quá trình có thể được mô tả như là quá trình hướng nhập/xuất (I/O-bound proces) hay quá trình hướng CPU (CPU-bound process). Một quá trình hướng nhập/xuất mất nhiều thời gian hơn để thực hiện nhập/xuất hơn thời gian tính toán. Ngược lại, một quá trình hướng CPU phát sinh các yêu cầu nhập/xuất không thường xuyên, dùng thời gian để thực hiện việc tính toán hơn một quá trình hướng nhập/xuất dùng. Bộ định thời công việc nên chọn sự kết hợp hài hoà giữa quá trình hướng nhập/xuất và quá trình hướng CPU. Nếu tất cả quá trình là hướng nhập/xuất thì hàng đợi sẵn sàng sẽ luôn rỗng và bộ định thời CPU sẽ có ít công việc để thực hiện. Nếu tất cả quá trình là hướng CPU thì hàng đợi nhập/xuất sẽ luôn rỗng, các thiết bị sẽ không được sử dụng và hệ thống sẽ mất cân bằng. Hệ thống với năng lực tốt nhất sẽ có sự kết hợp các quá trình hướng CPU và hướng nhập/xuất.

Trên một vài hệ thống, bộ định thời công việc có thể không có hay rất ít. Thí dụ, các hệ thống chia thời như UNIX thường không có bộ định thời công việc nhưng đơn giản đặt mỗi quá trình mới vào bộ nhớ cho bộ định thời CPU. Khả năng ổn định của hệ thống này phụ thuộc vào giới hạn vật lý (như số lượng thiết bị cuối sẵn dùng) hay tính tự nhiên tự chuyển đổi của người dùng. Nếu năng lực thực hiện giảm tới mức độ không thể chấp nhận được thì một số người dùng sẽ thoát khỏi hệ thống.

Một số hệ thống như hệ chia thời có thể đưa vào một cấp độ định thời bổ sung hay định thời trung gian. Bộ định thời trung gian (medium-term process) này (được hiển thị trong lưu đồ hình III-6) xóa các quá trình ra khỏi bộ nhớ (từ sự canh tranh CPU) và do đó giảm mức độ đa chương. Tại thời điểm sau đó, quá trình có thể được đưa trở lại bộ nhớ và việc thực thi của nó có thể được tiếp tục nơi nó bị đưa ra. Cơ chế này được gọi là hoán vị (swapping). Quá trình được hoán vị ra và sau đó được hoán vị vào bởi bộ định thời trung gian. Hoán vị là cần thiết để cải tiến sự trộn lẫn quá trình (giữa các quá trình hướng nhập/xuất và hướng CPU), hay vì một thay đổi trong yêu cầu bộ nhớ vượt quá kích thước bộ nhớ sẵn dùng. Hoán vị sẽ được thảo luận trong chương sau.



Hình III-6-Lưu đồ bổ sung định thời trung bình tới hàng đợi

Chuyển ngữ cảnh

Chuyển CPU tới một quá trình khác yêu cầu lưu trạng thái của quá trình cũ và nạp trạng thái được lưu cho quá trình mới. Tác vụ này được xem như chuyển ngữ cảnh (context switch). Ngữ cảnh của quá trình được hiện diện trong PCB của quá trình; Nó chứa giá trị các thanh ghi, trạng thái quá trình (hình III.1) và thông tin quản lý bộ nhớ. Khi chuyển ngữ cảnh ngữ cảnh xảy ra, nhân lưu ngữ cảnh của quá trình cũ trong PCB của nó và nạp ngữ cảnh được lưu của quá trình mới được định thời để chạy. Thời gian chuyển ngữ cảnh là chi phí thuần vì hệ thống không thực hiện công việc có ích trong khi chuyển. Tốc độ của nó khác từ máy này tới máy khác phụ thuộc vào tốc độ bộ nhớ, số lượng thanh ghi phải được chép và sự tồn tại của các chỉ thị đặc biệt (như chỉ thị để nạp và lưu tất cả thanh ghi). Điển hình dãy tốc độ từ 1 tới 1000 mili giây.

Những lần chuyển đổi ngữ cảnh phụ thuộc nhiều vào hỗ trợ phần cứng. Thí dụ, vài bộ xử lý (như Sun UltraSPARC) cung cấp nhiều tập thanh ghi. Một chuyển ngữ cảnh đơn giản chứa chuyển đổi con trỏ tới tập thanh ghi hiện hành. Dĩ nhiên, nếu quá trình hoạt động vượt quá tập thanh ghi thì hệ thống sắp xếp lại dữ liệu thanh ghi tới và từ bộ nhớ. Cũng vì thế mà hệ điều hành phức tạp hơn và nhiều công việc được làm hơn trong khi chuyển ngữ cảnh. Kỹ thuật quản lý bộ nhớ nâng cao có thể yêu cầu dữ liệu bổ sung để được chuyển với mỗi ngữ cảnh. Thí dụ, không gian địa chỉ của quá trình hiện hành phải được lưu khi không gian

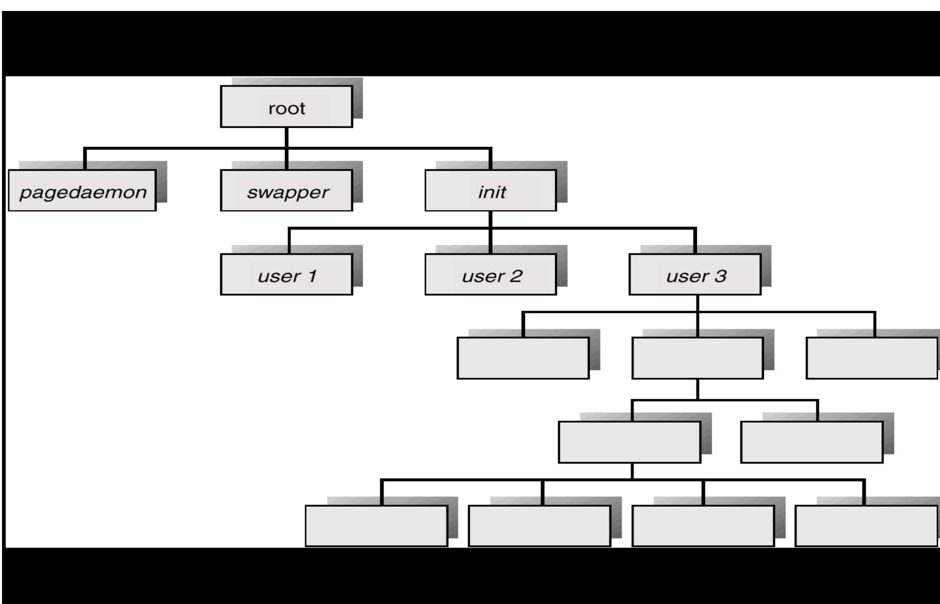
của tác vụ kế tiếp được chuẩn bị dùng. Không gian địa chỉ được lưu như thế nào và lượng công việc được yêu cầu để lưu nó phụ thuộc vào phương pháp quản lý bộ nhớ của hệ điều hành. Chuyển ngữ cảnh có thể dẫn đến thắt cổ chai năng lực thực hiện vì thế các lập trình viên đang sử dụng các cấu trúc mới để tránh nó bất cứ khi nào có thể.

Thao tác trên quá trình

Các quá trình trong hệ thống có thể thực thi đồng hành và chúng phải được tạo và xóa tự động. Do đó, hệ điều hành phải cung cấp một cơ chế (hay phương tiện) cho việc tạo quá trình và kết thúc quá trình.

Tạo quá trình

Quá trình có thể tạo nhiều quá trình mới, bằng một lời gọi hệ thống create-process, trong khi thực thi. Quá trình tạo gọi là quá trình cha, ngược lại các quá trình mới được gọi là quá trình con của quá trình đó. Mỗi quá trình mới này sau đó có thể tạo các quá trình khác, hình thành một cây quá trình (hình III-7).



Hình III-7-Cây quá trình trên một hệ thống UNIX điển hình

Thông thường, một quá trình sẽ cần các tài nguyên xác định (như thời gian CPU, bộ nhớ, tập tin, thiết bị nhập/xuất) để hoàn thành tác vụ của nó. Khi một quá trình tạo một quá trình con, quá trình con có thể nhận tài nguyên của nó trực tiếp từ hệ điều hành hay nó có thể bị ràng buộc tới một tập con các tài nguyên của quá trình cha. Quá trình cha có thể phải phân chia các tài nguyên giữa các quá trình con hay có thể chia sẻ một số tài nguyên (như bộ nhớ và tập tin) giữa nhiều quá trình con. Giới hạn một quá trình con tới một tập con tài nguyên của quá trình cha ngăn chặn bất cứ quá trình từ nạp chồng hệ thống bằng cách tạo quá nhiều quá trình con.

Ngoài ra, khi một quá trình được tạo nó lấy tài nguyên vật lý và luận lý khác nhau, dữ liệu khởi tạo (hay nhập) có thể được truyền từ quá trình cha tới quá trình con. Thí dụ, xét một quá trình với toàn bộ chức năng là hiển thị trạng thái của một tập tin F1 trên màn hình. Khi nó được tạo, nó sẽ lấy dữ liệu vào từ quá trình cha, tên của tập tin F1 và nó sẽ thực thi dùng dữ liệu đó để lấy thông tin mong muốn.

Nó có thể cũng lấy tên của thiết bị xuất. Một số hệ điều hành chuyển tài nguyên tới quá trình con. Trên hệ thống như thế, quá trình mới có thể lấy hai tập tin đang mở, F1 và thiết bị cuối, và có thể chỉ cần chuyển dữ liệu giữa hai tập tin.

Khi một quá trình tạo một quá trình mới, hai khả năng có thể tồn tại trong thuật ngữ của việc thực thi:

- Quá trình cha tiếp tục thực thi đồng hành với quá trình con của nó.
- Quá trình cha chờ cho tới khi một vài hay tất cả quá trình con kết thúc.

Cũng có hai khả năng trong thuật ngữ không gian địa chỉ của quá trình mới:

- Quá trình con là bản sao của quá trình cha.
- Quá trình con có một chương trình được nạp vào nó.

Để hiển thị việc cài đặt khác nhau này, chúng ta xem xét hệ điều hành UNIX. Trong UNIX, mỗi quá trình được xác định bởi danh biểu quá trình (process identifier), là số nguyên duy nhất. Một quá trình mới được tạo bởi lời gọi hệ thống fork. Quá trình mới chứa bản sao của không gian địa chỉ của quá trình gốc. Cơ chế này cho phép quá trình cha giao tiếp dễ dàng với quá trình con. Cả hai quá trình (cha và con) tiếp tục thực thi tại chỉ thị sau khi lời gọi hệ thống fork, với một sự khác biệt: mã trả về cho lời gọi hệ thống fork là không cho quá trình mới (con), ngược lại danh biểu quá trình (khác không) của quá trình con được trả về tới quá trình cha.

Điển hình lời gọi hệ thống execlp được dùng sau lời gọi hệ thống fork bởi một trong hai quá trình để thay thế không gian bộ nhớ với quá trình mới. Lời gọi hệ thống execlp nạp tập tin nhị phân vào trong bộ nhớ-xóa hình ảnh bộ nhớ của chương trình chứa lời gọi hệ thống execlp - và bắt đầu việc thực thi của nó. Trong cách thức này, hai quá trình có thể giao tiếp và sau đó thực hiện cách riêng của nó. Sau đó, quá trình cha có thể tạo nhiều hơn quá trình con, hay nếu nó không làm gì trong thời gian quá trình con chạy thì nó sẽ phát ra lời gọi hệ thống wait để di chuyển nó vào hàng đợi sẵn sàng cho tới khi quá trình con kết thúc. Chương trình C (hình III-8 dưới đây) hiển thị lời gọi hệ thống UNIX được mô tả trước đó. Quá trình cha tạo một quá trình con sử dụng lời gọi hệ thống fork. Bây giờ chúng ta có hai quá trình khác nhau chạy một bản sao của cùng chương trình. Giá trị pid cho quá trình con là 0; cho quá trình cha là một số nguyên lớn hơn 0. Quá trình con phủ lắp không gian địa chỉ của nó với lệnh của UNIX là /bin/ls (được dùng để liệt kê thư mục) dùng lời gọi hệ thống execlp. Quá trình cha chờ cho quá trình con hoàn thành với lời gọi hệ thống wait. Khi quá trình con hoàn thành, quá trình cha bắt đầu lại từ lời gọi hệ thống wait nơi nó hoàn thành việc sử dụng lời gọi hệ thống exit.

Ngược lại, hệ điều hành DEC VMS tạo một quá trình mới, nạp chương trình xác định trong quá trình đó và bắt đầu thực thi nó. Hệ điều hành Microsoft Windows NT hỗ trợ cả hai mô hình: không gian địa chỉ của quá trình cha có thể được sao lại hay quá trình cha có thể xác định tên của một chương trình cho hệ điều hành nạp vào không gian địa chỉ của quá trình mới.

```
#include<stdio.h>

main(int argc, char* argv[])

{

intpid;

/*fork another process*/

pid=fork();

if(pid<0){/*error occurred */

fprintf(stderr, “Fork Failed”);

exit(-1);

}

else if (pid==0){/*child process*/

execlp(“/bin/ls”,”ls”,NULL);

}

else {/*parent process*/

/*parent will wait for the child to complete*/

wait(NULL);

printf(“Child Complete”);

exit(0);

}

}
```

Hình III-8-Chương trình C tạo một quá trình riêng rẻ

Kết thúc quá trình

Một quá trình kết thúc khi nó hoàn thành việc thực thi câu lệnh cuối cùng và yêu cầu hệ điều hành xóa nó bằng cách sử dụng lời gọi hệ thống exit. Tại thời điểm đó, quá trình có thể trả về dữ liệu (đầu ra) tới quá trình cha (bằng lời gọi hệ thống wait). Tất cả tài nguyên của quá trình –gồm bộ nhớ vật lý và luận lý, các tập tin đang mở, vùng đệm nhập/xuất-được thu hồi bởi hệ điều hành.

Việc kết thúc xảy ra trong các trường hợp khác. Một quá trình có thể gây kết thúc một quá trình khác bằng một lời gọi hệ thống hợp lý (thí dụ, abort). Thường chỉ có quá trình cha bị kết thúc có thể gọi lời gọi hệ thống như thế. Ngược lại, người dùng có thể tùy ý giết (kill) mỗi công việc của quá trình còn lại. Do đó, quá trình cha cần biết các định danh của các quá trình con. Vì thế khi một quá trình tạo một quá trình mới, định danh của mỗi quá trình mới được tạo được truyền tới cho quá trình cha.

Một quá trình cha có thể kết thúc việc thực thi của một trong những quá trình con với nhiều lý do khác nhau:

- Quá trình con sử dụng tài nguyên vượt quá mức được cấp. Điều này yêu cầu quá trình cha có một cơ chế để xem xét trạng thái của các quá trình con.
- Công việc được gán tới quá trình con không còn cần thiết nữa.
- Quá trình cha đang kết thúc và hệ điều hành không cho phép một quá trình con tiếp tục nếu quá trình cha kết thúc. Trên những hệ thống như thế, nếu một quá trình kết thúc (bình thường hoặc không bình thường), thì tất cả quá trình con cũng phải kết thúc. Trường hợp này được xem như kết thúc xếp tầng (cascading termination) thường được khởi tạo bởi hệ điều hành.

Để hiển thị việc thực thi và kết thúc quá trình, xem xét hệ điều hành UNIX chúng ta có thể kết thúc một quá trình dùng lời gọi hệ thống exit; nếu quá trình cha có thể chờ cho đến khi quá trình con kết thúc bằng lời gọi hệ thống wait. Lời gọi hệ thống wait trả về định danh của quá trình con bị kết thúc để quá trình cha có thể xác định những quá trình con nào có thể kết thúc. Tuy nhiên, nếu quá trình cha kết thúc thì tất cả quá trình con của nó được gán như quá trình cha mới của chúng, quá trình khởi tạo (init process). Do đó, các quá trình con chỉ có một quá trình cha để tập hợp trạng thái và thống kê việc thực thi.

Hợp tác quá trình

Các quá trình đồng hành thực thi trong hệ điều hành có thể là những quá trình độc lập hay những quá trình hợp tác. Một quá trình là độc lập (independent) nếu nó không thể ảnh hưởng hay bị ảnh hưởng bởi các quá trình khác thực thi trong hệ thống. Rõ ràng, bất kỳ một quá trình không chia sẻ bất cứ dữ liệu (tạm thời hay cố định) với quá trình khác là độc lập. Ngược lại, một quá trình là hợp tác (cooperating) nếu nó có thể ảnh hưởng hay bị ảnh hưởng bởi các quá trình khác trong hệ thống. Hay nói cách khác, bất cứ quá trình chia sẻ dữ liệu với quá trình khác là quá trình hợp tác.

Chúng ta có thể cung cấp một môi trường cho phép hợp tác quá trình với nhiều lý do:

- Chia sẻ thông tin: vì nhiều người dùng có thể quan tâm cùng phần thông tin (thí dụ, tập tin chia sẻ), chúng phải cung cấp một môi trường cho phép truy xuất đồng hành tới những loại tài nguyên này.
- Gia tăng tốc độ tính toán: nếu chúng ta muốn một tác vụ chạy nhanh hơn, chúng ta phải chia nó thành những tác vụ nhỏ hơn, mỗi tác vụ sẽ thực thi song song với các tác vụ khác. Việc tăng tốc như thế có thể đạt được chỉ nếu máy tính có nhiều thành phần đa xử lý (như các CPU hay các kênh I/O).
- Tính module hóa: chúng ta muốn xây dựng hệ thống trong một kiểu mẫu dạng module, chia các chức năng hệ thống thành những quá

trình hay luồng như đã thảo luận ở chương trước.
- Tính tiện dụng: Thậm chí một người dùng đơn có thể có nhiều tác vụ thực hiện tại cùng thời điểm. Thí dụ, một người dùng có thể đang soạn thảo, in, và biên dịch cùng một lúc.

Thực thi đồng hành của các quá trình hợp tác yêu cầu các cơ chế cho phép các quá trình giao tiếp với các quá trình khác và đồng bộ hóa các hoạt động của chúng.

Để minh họa khái niệm của các quá trình cộng tác, chúng ta xem xét bài toán người sản xuất-người tiêu thụ, là mô hình chung cho các quá trình hợp tác. Quá trình người sản xuất cung cấp thông tin được tiêu thụ bởi quá trình người tiêu thụ. Thí dụ, một chương trình in sản xuất các ký tự được tiêu thụ bởi trình điều khiển máy in. Một trình biên dịch có thể sản xuất mã hợp ngữ được tiêu thụ bởi trình hợp ngữ. Sau đó, trình hợp ngữ có sản xuất module đối tượng, được tiêu thụ bởi bộ nạp.

Để cho phép người sản xuất và người tiêu thụ chạy đồng hành, chúng ta phải có sẵn một vùng đệm chứa các sản phẩm có thể được điền vào bởi người sản xuất và được lấy đi bởi người tiêu thụ. Người sản xuất có thể sản xuất một sản phẩm trong khi người tiêu thụ đang tiêu thụ một sản phẩm khác. Người sản xuất và người tiêu thụ phải được đồng bộ để mà người tiêu thụ không cố gắng tiêu thụ một sản phẩm mà chưa được sản xuất. Trong trường hợp này, người tiêu thụ phải chờ cho tới khi các sản phẩm mới được tạo ra.

Bài toán người sản xuất-người tiêu thụ với vùng đệm không bị giới hạn (unbounded-buffer) thiết lập không giới hạn kích thước của vùng đệm. Người tiêu thụ có thể phải chờ các sản phẩm mới nhưng người sản xuất có thể luôn tạo ra sản phẩm mới. Vấn đề người sản xuất-người tiêu thụ với vùng đệm có kích thước giới hạn (bounded-buffer) đảm bảo một kích thước cố định cho vùng đệm. Trong trường hợp này, người tiêu thụ phải chờ nếu vùng đệm rỗng, và người sản xuất phải chờ nếu vùng đệm đầy.

Vùng đệm có thể được cung cấp bởi hệ điều hành thông qua việc sử dụng phương tiện giao tiếp liên quá trình (interprocess-communication-

IPC), hay được mã hóa cụ thể bởi người lập trình ứng dụng với việc sử dụng bộ nhớ được chia sẻ. Để chúng ta hiển thị một giải pháp chia sẻ bộ nhớ đối với vấn đề vùng đệm bị giới hạn (bounded-buffer). Quá trình người sản xuất và người tiêu thụ chia sẻ các biến sau:

```
#define BUFFER_SIZE10

typedefstruct{

…

} item;

item buffer[BUFFER_SIZE];

intin=0;

intout=0;
```

Vùng đệm được chia sẻ được cài đặt như một mảng vòng với hai con trỏ luận lý: in và out. Biến in chỉ tới vị trí trống kế tiếp trong vùng đệm; out chỉ tới vị trí đầy đầu tiên trong vùng đệm. Vùng đệm là rỗng khi in==out; vùng đệm là đầy khi ((in + 1)%BUFFER_SIZE) ==out.

Mã cho quá trình người sản xuất và người tiêu thụ được trình bày dưới đây. Quá trình người sản xuất có một biến nextProduced trong đó sản phẩm mới được tạo ra và được lưu trữ:

```
while (1) {

/*produce an item in nextProduced*/

while (((in + 1)%BUFFER_SIZE) ==out)

; /*do nothing*/

buffer[in]=nextProduced;

in=(in + 1) % BUFFER_SIZE;
```

```
}
```

Quá trình người tiêu thụ có biến cục bộ nextConsumed trong đó sản phẩm được tiêu thụ và được lưu trữ:

```
while (1){

while (in==out)

; //nothing

nextConsumed=buffer[out];

out=(out+1)% BUFFER_SIZE;

/*consume the item in nextConsumed*/

}
```

Cơ chế này cho phép nhiều nhất BUFFER_SIZE –1 trong vùng đệm tại cùng một thời điểm.

Giao tiếp liên quá trình

Trong phần trên chúng ta đã hiển thị cách mà các quá trình hợp tác có thể giao tiếp với nhau trong một môi trường chia sẻ bộ nhớ. Cơ chế yêu cầu các quá trình này chia sẻ nhóm vùng đệm chung và mã cho việc cài đặt vùng đệm được viết trực tiếp bởi người lập trình ứng dụng. Một cách khác đạt được cùng ảnh hưởng cho hệ điều hành là cung cấp phương tiện cho các quá trình hợp tác giao tiếp với nhau bằng một phương tiện giao tiếp liên quá trình (IPC).

IPC cung cấp một cơ chế cho phép một quá trình giao tiếp và đồng bộ các hoạt động của chúng mà không chia sẻ cùng không gian địa chỉ. IPC đặc biệt có ích trong môi trường phân tán nơi các quá trình giao tiếp có thể thường trú trên các máy tính khác được nối kết qua mạng. Thí dụ chương trình chat được dùng trên World Wide Web.

IPC được cung cấp bởi hệ thống truyền thông điệp, và các hệ thống truyền thông điệp có thể được định nghĩa trong nhiều cách. Trong phần này chúng ta sẽ xem xét những vấn đề khác nhau khi thiết kế các hệ thống truyền thông điệp.

Hệ thống truyền thông điệp

Chức năng của hệ thống truyền thông điệp là cho phép các quá trình giao tiếp với các quá trình khác mà không cần sắp xếp lại dữ liệu chia sẻ. Chúng ta xem truyền thông điệp được dùng như một phương pháp giao tiếp trong vi nhân. Trong cơ chế này, các dịch vụ được cung cấp như các quá trình người dùng thông thường. Nghĩa là, các dịch vụ hoạt động bên ngoài nhân. Giao tiếp giữa các quá trình người dùng được thực hiện thông qua truyền thông điệp. Một phương tiện IPC cung cấp ít nhất hai hoạt động: send(message) và receive(message).

Các thông điệp được gởi bởi một quá trình có thể có kích thước cố định hoặc biến đổi. Nếu chỉ các thông điệp có kích thước cố định được gởi, việc cài đặt cấp hệ thống là đơn giản hơn. Tuy nhiên, hạn chế này làm cho tác vụ lập trình sẽ phức tạp hơn. Ngoài ra, các thông điệp có kích thước thay đổi yêu cầu việc cài đặt mức hệ thống phức tạp hơn nhưng tác vụ lập trình trở nên đơn giản hơn.

Nếu quá trình P và Q muốn giao tiếp, chúng phải gởi các thông điệp tới và nhận thông điệp từ với nhau; một liên kết giao tiếp phải tồn tại giữa chúng. Liên kết này có thể được cài đặt trong những cách khác nhau. Ở đây chúng ta quan tâm đến cài đặt luận lý hơn là cài đặt vật lý. Có vài phương pháp cài đặt một liên kết và các hoạt động send/receive:

- Giao tiếp trực tiếp hay gián tiếp
- Giao tiếp đối xứng hay bất đối xứng
- Gởi bằng bản sao hay tham chiếu
- Thông điệp có kích thước cố định hay thay đổi

Đặt tên

Các quá trình muốn giao tiếp phải có cách tham chiếu với nhau. Chúng có thể dùng giao tiếp trực tiếp hay gián tiếp.

Giao tiếp trực tiếp

Với giao tiếp trực tiếp, mỗi quá trình muốn giao tiếp phải đặt tên rõ ràng người gởi và người nhận của giao tiếp. Trong cơ chế này, các hàm cơ sở send và receive được định nghĩa như sau:

- Send(P, message):gởi một thông điệp tới quá trình P
- Receive(Q, message):nhận một thông điệp từ quá trình Q

Một liên kết giao tiếp trong cơ chế này có những thuộc tính sau:

- Một liên kết được thiết lập tự động giữa mỗi cặp quá trình muốn giao tiếp. Các quá trình cần biết định danh của nhau khi giao tiếp.
- Một liên kết được nối kết với chính xác hai quá trình
- Chính xác một liên kết tồn tại giữa mỗi cặp quá trình.

Cơ chế này hiển thị tính đối xứng trong việc đánh địa chỉ: nghĩa là, cả hai quá trình gởi và nhận phải biết tên nhau để giao tiếp. Một thay đổi trong cơ chế này thực hiện tính bất đối xứng trong việc đánh địa chỉ. Chỉ người gởi biết tên của người nhận; người nhận không yêu cầu tên của người gởi. Trong cơ chế này các hàm cơ sở được định nghĩa như sau:

- Send(P, message):gởi một thông điệp tới quá trình P
- Receive(id, message):nhận một thông điệp từ bất kỳ quá trình nào; id khác nhau được đặt tên của quá trình mà giao tiếp xảy ra.

Sự bất lợi trong cả hai cơ chế đối xứng và không đối xứng là tính điều chỉnh của việc định nghĩa quá trình bị giới hạn. Thay đổi tên của một quá trình có thể cần xem xét tất cả định nghĩa quá trình khác. Tất cả tham chiếu tới tên cũ phải được tìm thấy để mà chúng có thể được thay đổi thành tên mới. Trường hợp này là không mong muốn từ quan điểm biên dịch riêng.

Giao tiếp gián tiếp

Với giao tiếp gián tiếp, một thông điệp được gởi tới và nhận từ các hộp thư (mailboxes), hay cổng (ports). Một hộp thư có thể được hiển thị trừu tượng như một đối tượng trong đó các thông điệp có thể được đặt bởi các quá trình và sau đó các thông điệp này có thể được xóa đi. Mỗi hộp thư có một định danh duy nhất. Trong cơ chế này, một quá trình có thể giao tiếp với một vài quá trình khác bằng một số hộp thư khác nhau. Hai quá trình có thể giao tiếp chỉ nếu chúng chia sẻ cùng một hộp thư. Hàm cơ sở send và receive được định nghĩa như sau:

- Send(A, message):gởi một thông điệp tới hộp thư A.
- Receive(A, message):nhận một thông điệp từ hộp thư A.

Trong cơ chế này, một liên kết giao tiếp có các thuộc tính sau:

- Một liên kết được thiết lập giữa một cặp quá trình chỉ nếu cả hai thành viên của cặp có một hộp thư được chia sẻ.
- Một liên kết có thể được nối kết với nhiều hơn hai quá trình.
- Số các liên kết khác nhau có thể tồn tại giữa mỗi cặp quá trình giao tiếp với mỗi liên kết tương ứng với một hộp thư

Giả sử các quá trình P1, P2 và P3 chia sẻ một hộp thư A. Quá trình P1 gởi một thông điệp tới A trong khi P2 và P3 thực thi việc nhận từ A. Quá trình nào sẽ nhận thông điệp được gởi bởi P1? Câu trả lời phụ thuộc cơ chế mà chúng ta chọn:

- Cho phép một liên kết được nối kết với nhiều nhất hai quá trình
- Cho phép nhiều nhất một quá trình tại một thời điểm thực thi thao tác nhận.
- Cho phép hệ thống chọn bất kỳ quá trình nào sẽ nhận thông điệp (nghĩa là, hoặc P1 hoặc P3 nhưng không phải cả hai sẽ nhận thông điệp). Hệ thống này có thể xác định người nhận tới người gởi.

Một hộp thư có thể được sở hữu bởi một quá trình hay bởi hệ điều hành. Nếu hộp thư được sở hữu bởi một quá trình (nghĩa là, hộp thư là một phần không gian địa chỉ của quá trình), sau đó chúng ta phân biệt giữa

người sở hữu (người chỉ nhận thông điệp thông qua hộp thư này) và người dùng (người có thể chỉ gởi thông điệp tới hộp thư). Vì mỗi hộp thư có một người sở hữu duy nhất nên không có sự lẫn lộn về người nhận thông điệp được gởi tới hộp thư này. Khi một quá trình sở hữu một hộp thư kết thúc, hộp thư biến mất. Sau đó, bất kỳ quá trình nào gởi thông điệp tới hộp thư này được thông báo rằng hộp thư không còn tồn tại nữa.

Ngoài ra, một hộp thư được sở hữu bởi hệ điều hành độc lập và không được gán tới bất kỳ quá trình xác định nào. Sau đó, hệ điều hành phải cung cấp một cơ chế cho phép một quá trình thực hiện như sau:

- Tạo một hộp thư mới
- Gởi và nhận các thông điệp thông qua hộp thư
- Xóa hộp thư

Mặc định, quá trình tạo hộp thư mới là người sở hữu hộp thư đó. Ban đầu, người sở hữu chỉ là một quá trình có thể nhận thông điệp thông qua hộp thư. Tuy nhiên, việc sở hữu và quyền nhận thông điệp có thể được chuyển tới các quá trình khác thông qua lời gọi hệ thống hợp lý. Dĩ nhiên, sự cung cấp này có thể dẫn đến nhiều người nhận cho mỗi hộp thư.

Đồng bộ hóa

Giao tiếp giữa hai quá trình xảy ra bởi lời gọi hàm cơ sở send và receive. Có các tùy chọn thiết kế khác nhau cho việc cài đặt mỗi hàm cơ sở. Truyền thông điệp có thể là nghẽn (block) hay không nghẽn (nonblocking)-cũng được xem như đồng bộ và bất đồng bộ.

- Hàm send nghẽn: quá trình gởi bị nghẽn cho đến khi thông điệp được nhận bởi quá trình nhận hay bởi hộp thư.
- Hàm send không nghẽn: quá trình gởi gởi thông điệp và thực hiện tiếp hoạt động
- Hàm receive nghẽn: người nhận nghẽn cho đến khi thông điệp sẵn dùng
- Hàm receive không nghẽn: người nhận nhận lại một thông điệp hợp lệ hay rỗng

Sự kết hợp khác nhau giữa send và receive là có thể. Khi cả hai send và receive là nghẽn chúng ta có sự thống nhất giữa người gởi và người nhận.

Tạo vùng đệm

Dù giao tiếp có thể là trực tiếp hay gián tiếp, các thông điệp được chuyển đổi bởi các quá trình giao tiếp thường trú trong một hàng đợt tạm thời. Về cơ bản, một hàng đợi như thế có thể được cài đặt trong ba cách:

- Khả năng chứa là 0 (zero capacity): hàng đợi có chiều dài tối đa là 0; do đó liên kết không thể có bất kỳ thông điệp nào chờ trong nó. Trong trường hợp này, người gởi phải nghẽn cho tới khi người nhận nhận thông điệp.
- Khả năng chứa có giới hạn (bounded capacity): hàng đợi có chiều dài giới hạn n; do đó, nhiều nhất n thông điệp có thể thường trú trong nó. Nếu hàng đợi không đầy khi một thông điệp mới được gởi, sau đó nó được đặt trong hàng đợi (thông điệp được chép hay một con trỏ thông điệp được giữ) và người gởi có thể tiếp tục thực thi không phải chờ. Tuy nhiên, liên kết có khả năng chứa giới hạn. Nếu một liên kết đầy, người gởi phải nghẽn cho tới khi không gian là sẵn dùng trong hàng đợi
- Khả năng chứa không giới hạn (unbounded capacity): Hàng đợi có khả năng có chiều dài không giới hạn; do đó số lượng thông điệp bất kỳ có thể chờ trong nó. Người gởi không bao giờ nghẽn.

Trường hợp khả năng chứa là 0 thường được xem như hệ thống thông điệp không có vùng đệm; hai trường hợp còn lại được xem như vùng đệm tự động.

Tóm tắt

Quá trình là một chương trình đang thực thi. Khi một quá trình thực thi, nó thay đổi trạng thái. Trạng thái của một quá trình được định nghĩa bởi một hoạt động hiện tại của quá trình đó. Mỗi quá trình có thể ở một trong những trạng thái sau: mới (new), sẵn sàng (ready), đang chạy

(running), chờ (waiting), hay kết thúc (terminated). Mỗi quá trình được biểu diễn trong hệ điều hành bởi khối điều khiển quá trình của chính nó (PCB).

Một quá trình khi không thưc thi, được đặt vào hàng đợi. Hai cấp chủ yếu của hàng đợi trong hệ điều hành là hàng đợi yêu cầu nhập/xuất và hàng đợi sẳn sàng. Hàng đợi sẳn sàng chứa tất cả quá trình sẳn sàng để thực thi và đang chờ CPU. Mỗi quá trình được biểu diễn bởi một PCB và các PCB có thể được liên kết với nhau để hình thành một hàng đợi sẳn sàng. Định thời biểu dài (long-term scheduling) (hay định thời biểu công việc) là chọn các quá trình được phép cạnh tranh CPU. Thông thường, định thời biểu dài bị ảnh hưởng nặng nề bởi việc xem xét cấp phát tài nguyên, đặc biệt quản lý bộ nhớ. Định thời ngắn (short-term scheduling) là sự chọn lựa một quá trình từ các hàng đợi sẳn sàng.

Các quá trình trong hệ thống có thể thực thi đồng hành. Có nhiều lý do các thực thi đồng hành: chia sẻ thông tin, tăng tốc độ tính toán, hiệu chỉnh và tiện dụng. Thực thi đồng hành yêu cầu cơ chế cho việc tạo và xóa quá trình.

Quá trình thực thi trong hệ điều hành có thể là các quá trình độc lập hay các quá trình hợp tác. Các quá trình hợp tác phải có phương tiện giao tiếp với nhau. Chủ yếu, có hai cơ chế giao tiếp bổ sung cho nhau cùng tồn tại: chia sẻ bộ nhớ và hệ thống truyền thông điệp. Phương pháp chia sẻ bộ nhớ yêu cầu các quá trình giao tiếp chia sẻ một số biến. Các quá trình được mong đợi trao đổi thông tin thông qua việc sử dụng các biến dùng chung này. Trong hệ thống bộ nhớ được chia sẻ, nhiệm vụ cho việc cung cấp giao tiếp tách rời với người lập trình ứng dụng; chỉ hệ điều hành cung cấp hệ thống bộ nhớ được chia sẻ. Phương pháp truyền thông điệp cho phép các quá trình trong đổi thông điệp. Nhiệm vụ cung cấp giao tiếp có thể tách rời với hệ điều hành. Hai cơ chế này không loại trừ hỗ tương và có thể được dùng cùng một lúc trong phạm vi một hệ điều hành.

Định thời biểu CPU
1 Mục tiêu Sau khi học xong chương này, người học nắm được những kiến thức sau: - Hiểu các khái niệm cơ bản về định thời - Hiểu các giải thuật định thời biểu CPU - Vận dụng một giải thuật định thời cho một hệ thống cụ thể

Giới thiệu

Định thời biểu là cơ sở của các hệ điều hành đa chương. Bằng cách chuyển đổi CPU giữa các quá trình, hệ điều hành có thể làm máy tính hoạt động nhiều hơn. Trong chương này, chúng ta giới thiệu các khái niệm định thời cơ bản và trình bày các giải thuật định thời biểu CPU khác nhau. Chúng ta cũng xem xét vấn đề chọn một giải thuật cho một hệ thống xác định.

Các khái niệm cơ bản

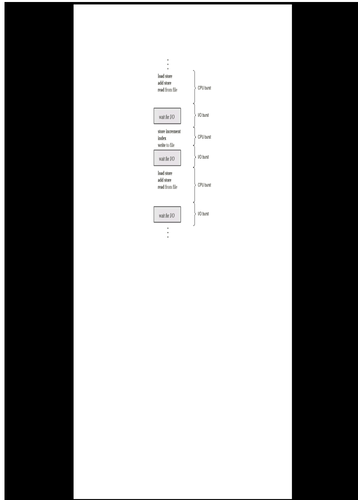
Mục tiêu của đa chương là có nhiều quá trình chạy cùng thời điểm để tối ưu hóa việc sử dụng CPU. Trong hệ thống đơn xử lý, chỉ một quá trình có thể chạy tại một thời điểm; bất cứ quá trình nào khác đều phải chờ cho đến khi CPU rảnh và có thể được định thời lại.

Ý tưởng của đa chương là tương đối đơn giản. Một quá trình được thực thi cho đến khi nó phải chờ yêu cầu nhập/xuất hoàn thành. Trong một hệ thống máy tính đơn giản thì CPU sẽ rảnh rỗi; tất cả thời gian chờ này là lãng phí. Với đa chương, chúng ta cố gắng dùng thời gian này để CPU có thể phục vụ cho các quá trình khác. Nhiều quá trình được giữ trong bộ nhớ tại cùng thời điểm. Khi một quá trình phải chờ, hệ điều hành lấy CPU từ quá trình này và cấp CPU tới quá trình khác.

Định thời biểu là chức năng cơ bản của hệ điều hành. Hầu hết tài nguyên máy tính được định thời biểu trước khi dùng. Dĩ nhiên, CPU là một trong những tài nguyên máy tính ưu tiên. Do đó, định thời biểu là trọng tâm trong việc thiết kế hệ điều hành.

Chu kỳ CPU-I/O

Sự thành công của việc định thời biểu CPU phụ thuộc vào thuộc tính được xem xét sau đây của quá trình. Việc thực thi quá trình chứa một chu kỳ (cycle) thực thi CPU và chờ đợi nhập/xuất. Các quá trình chuyển đổi giữa hai trạng thái này. Sự thực thi quá trình bắt đầu với một chu kỳ CPU (CPU burst), theo sau bởi một chu kỳ nhập/xuất (I/O burst), sau đó một chu kỳ CPU khác, sau đó lại tới một chu kỳ nhập/xuất khác khác,..Sau cùng, chu kỳ CPU cuối cùng sẽ kết thúc với một yêu cầu hệ thống để kết thúc việc thực thi, hơn là với một chu kỳ nhập/xuất khác, được mô tả như hình IV.1. Một chương trình hướng nhập/xuất (I/O-bound) thường có nhiều chu kỳ CPU ngắn. Một chương trình hướng xử lý (CPU-bound) có thể có một nhiều chu kỳ CPU dài. Sự phân bổ này có thể giúp chúng ta chọn giải thuật định thời CPU hợp lý.



Hình IV-1-Thay đổi thứ tự của CPU và I/O burst

Bộ định thời CPU

Bất cứ khi nào CPU rảnh, hệ điều hành phải chọn một trong những quá trình trong hàng đợi sẵn sàng để thực thi. Chọn quá trình được thực hiện bởi bộ định thời biểu ngắn (short-term scheduler) hay bộ định thời CPU. Bộ định thời này chọn các quá trình trong bộ nhớ sẵn sàng thực thi và cấp phát CPU tới một trong các quá trình đó.

Hàng đợi sẵn sàng không nhất thiết là hàng đợi vào trước, ra trước (FIFO). Xem xét một số giải thuật định thời khác nhau, một hàng đợi sẵn sàng có thể được cài đặt như một hàng đợi FIFO, một hàng đợi ưu tiên, một cây, hay đơn giản là một danh sách liên kết không thứ tự. Tuy nhiên, về khái niệm tất cả các quá trình trong hàng đợi sẵn sàng được xếp hàng chờ cơ hội để chạy trên CPU. Các mẫu tin trong hàng đợi thường là khối điều khiển quá trình của quá trình đó.

Định thời biểu trưng dụng

Quyết định định thời biểu CPU có thể xảy ra một trong 4 trường hợp sau:

1. Khi một quá trình chuyển từ trạng thái chạy sang trạng thái chờ (thí dụ: yêu cầu nhập/xuất, hay chờ kết thúc của một trong những quá trình con).
2. Khi một quá trình chuyển từ trạng thái chạy tới trạng thái sẵn sàng (thí dụ: khi một ngắt xảy ra)
3. Khi một quá trình chuyển từ trạng thái chờ tới trạng thái sẵn sàng (thí dụ: hoàn thành nhập/xuất)
4. Khi một quá trình kết thúc

Trong trường hợp 1 và 4, không cần chọn lựa loại định thời biểu. Một quá trình mới (nếu tồn tại trong hàng đợi sẵn sàng) phải được chọn để thực thi. Tuy nhiên, có sự lựa chọn loại định thời biểu trong trường hợp 2 và 3.

Khi định thời biểu xảy ra chỉ trong trường hợp 1 và 4, chúng ta nói cơ chế định thời không trưng dụng (nonpreemptive); ngược lại, khi định thời biểu xảy ra chỉ trong trường hợp 2 và 3, chúng ta nói cơ chế định thời trưng dụng (preemptive). Trong định thời không trưng dụng, một khi CPU được cấp phát tới một quá trình, quá trình giữ CPU cho tới khi nó giải phóng CPU hay bởi kết thúc hay bởi chuyển tới trạng thái sẵn sàng. Phương pháp định thời biểu này được dùng bởi các hệ điều hành Microsoft Windows 3.1 và bởi Apple Macintosh. Phương pháp này chỉ có thể được dùng trên các nền tảng phần cứng xác định vì nó không đòi hỏi phần cứng đặc biệt (thí dụ, một bộ đếm thời gian) được yêu cầu để định thời biểu trưng dụng.

Tuy nhiên, định thời trưng dụng sinh ra một chi phí. Xét trường hợp 2 quá trình chia sẻ dữ liệu. Một quá trình có thể ở giữa giai đoạn cập nhật dữ liệu thì nó bị chiếm dụng CPU và một quá trình thứ hai đang chạy. Quá trình thứ hai có thể đọc dữ liệu mà nó hiện đang ở trong trạng thái thay đổi. Do đó, những kỹ thuật mới được yêu cầu để điều phối việc truy xuất tới dữ liệu được chia sẻ.

Sự trưng dụng cũng có một ảnh hưởng trong thiết kế nhân hệ điều hành. Trong khi xử lý lời gọi hệ thống, nhân có thể chờ một hoạt động dựa theo hành vi của quá trình. Những hoạt động như thế có thể liên quan với sự thay đổi dữ liệu nhân quan trọng (thí dụ: các hàng đợi nhập/xuất). Điều gì xảy ra nếu quá trình bị trưng dụng CPU ở trong giai đoạn thay đổi này và nhân (hay trình điều khiển thiết bị) cần đọc hay sửa đổi cùng cấu trúc? Sự lộn xộn chắc chắn xảy ra. Một số hệ điều hành, gồm hầu hết các ấn bản của UNIX, giải quyết vấn đề này bằng cách chờ lời gọi hệ thống hoàn thành hay việc nhập/xuất bị nghẽn, trước khi chuyển đổi ngữ cảnh. Cơ chế này đảm bảo rằng cấu trúc nhân là đơn giản vì nhân sẽ không trưng dụng một quá trình trong khi các cấu trúc dữ liệu nhân ở trong trạng thái thay đổi. Tuy nhiên, mô hình thực thi nhân này là mô hình nghèo nàn để hỗ trợ tính toán thời thực và đa xử lý.

Trong trường hợp UNIX, các phần mã vẫn là sự rủi ro. Vì các ngắt có thể xảy ra bất cứ lúc nào và vì các ngắt này không thể luôn được bỏ qua bởi nhân, nên phần mã bị ảnh hưởng bởi ngắt phải được đảm bảo từ việc sử dụng đồng thời. Hệ điều hành cần chấp nhận hầu hết các ngắt, ngược lại dữ liệu nhập có thể bị mất hay dữ liệu xuất bị viết chồng. Vì thế các phần mã này không thể được truy xuất đồng hành bởi nhiều quá trình, chúng vô hiệu hóa ngắt tại lúc nhập và cho phép các ngắt hoạt động trở lại tại thời điểm việc nhập kết thúc. Tuy nhiên, vô hiệu hóa và cho phép ngắt tiêu tốn thời gian, đặc biệt trên các hệ thống đa xử lý.

Bộ phân phát

Một thành phần khác liên quan đến chức năng định thời biểu CPU là bộ phân phát (dispatcher). Bộ phân phát là một module có nhiệm vụ trao điều khiển CPU tới quá trình được chọn bởi bộ định thời biểu ngắn (short-term scheduler). Chức năng này liên quan:

- Chuyển ngữ cảnh
- Chuyển chế độ người dùng
- Nhảy tới vị trí hợp lý trong chương trình người dùng để khởi động lại quá trình

Bộ phân phát nên nhanh nhất có thể, và nó được nạp trong mỗi lần chuyển quá trình. Thời gian mất cho bộ phân phát dừng một quá trình này và bắt đầu chạy một quá trình khác được gọi là thời gian trễ cho việc điều phối (dispatch latency).

Các tiêu chuẩn định thời

Các giải thuật định thời khác nhau có các thuộc tính khác nhau và có xu hướng thiên vị cho một loại quá trình hơn một quá trình. Trong việc chọn giải thuật nào sử dụng trong trường hợp nào, chúng ta phải xét các thuộc tính của các giải thuật khác nhau.

Nhiều tiêu chuẩn được đề nghị để so sánh các giải thuật định thời biểu. Những đặc điểm được dùng để so sánh có thể tạo sự khác biệt quan trọng trong việc xác định giải thuật tốt nhất. Các tiêu chuẩn gồm:

- Việc sử dụng CPU: chúng ta muốn giữ CPU bận nhiều nhất có thể. Việc sử dụng CPU có thể từ 0 đến 100%. Trong hệ thống thực, nó nên nằm trong khoảng từ 40% (cho hệ thống được nạp tải nhẹ) tới 90% (cho hệ thống được nạp tải nặng).
- Thông lượng: nếu CPU bận thực thi các quá trình thì công việc đang được thực hiện. Thước đo của công việc là số lượng quá trình được hoàn thành trên một đơn vị thời gian gọi là thông lượng (throughput). Đối với các quá trình dài, tỉ lệ này có thể là 1 quá trình trên 1 giờ; đối với các giao dịch ngắn, thông lượng có thể là 10 quá trình trên giây.

- Thời gian hoàn thành: từ quan điểm của một quá trình cụ thể, tiêu chuẩn quan trọng là mất bao lâu để thực thi quá trình đó. Khoảng thời gian từ thời điểm gởi quá trình tới khi quá trình hoàn thành được gọi là thời gian hoàn thành (turnaround time). Thời gian hoàn thành là tổng các thời gian chờ đưa quá trình vào bộ nhớ, chờ hàng đợi sẵn sàng, thực thi CPU và thực hiện nhập/xuất.
- Thời gian chờ: giải thuật định thời CPU không ảnh hưởng lượng thời gian quá trình thực thi hay thực hiện nhập/xuất; nó ảnh hưởng chỉ lượng thời gian một quá trình phải chờ trong hàng đợi sẵn sàng. Thời gian chờ (waiting time) là tổng thời gian chờ trong hàng đợi sẵn sàng.
- Thời gian đáp ứng: trong một hệ thống giao tiếp, thời gian hoàn thành không là tiêu chuẩn tốt nhất. Thông thường, một quá trình có thể tạo ra dữ liệu xuất tương đối sớm và có thể tiếp tục tính toán các kết quả mới trong khi các kết quả trước đó đang được xuất cho người dùng. Do đó, một thước đo khác là thời gian từ lúc gởi yêu cầu cho tới khi đáp ứng đầu tiên được tạo ra. Thước đo này được gọi là thời gian đáp ứng (response time), là lượng thời gian mất đi từ lúc bắt đầu đáp ứng nhưng không là thời gian mất đi để xuất ra đáp ứng đó. Thời gian hoàn thành thường bị giới hạn bởi tốc độ của thiết bị xuất.

Chúng ta muốn tối ưu hóa việc sử dụng CPU và thông lượng, đồng thời tối thiểu hóa thời gian hoàn thành, thời gian chờ, và thời gian đáp ứng. Trong hầu hết các trường hợp, chúng ta tối ưu hóa thước đo trung bình. Tuy nhiên, trong một vài trường hợp chúng ta muốn tối ưu giá trị tối thiểu hay giá trị tối đa hơn là giá trị trung bình. Thí dụ, để đảm bảo rằng tất cả người dùng nhận dịch vụ tốt, chúng ta muốn tối thiểu thời gian đáp ứng tối đa.

Đối với những hệ thống tương tác (như các hệ thống chia thời), một số nhà phân tích đề nghị rằng sự thay đổi trong thời gian đáp ứng quan trọng hơn tối thiểu hóa thời gian đáp ứng trung bình. Một hệ thống với thời gian đáp ứng phù hợp và có thể đoán trước được quan tâm nhiều hơn hệ thống chạy nhanh hơn mức trung bình nhưng biến đổi cao. Tuy nhiên, gần như không có công việc nào được thực hiện trên các giải thuật định thời biểu CPU để tối thiểu hóa các thay đổi.

Khi chúng ta thảo luận các giải thuật định thời biểu CPU khác nhau, chúng ta muốn hiển thị các hoạt động của chúng. Một hình ảnh chính xác nên thông báo tới nhiều quá trình, mỗi quá trình là một chuỗi của hàng trăm chu kỳ CPU và I/O. Để đơn giản việc hiển thị, chúng ta xem chỉ một chu kỳ CPU (trong mili giây) trên quá trình trong các thí dụ của chúng ta. Thước đo của việc so sánh là thời gian chờ đợi trung bình.

Các giải thuật định thời

Định thời biểu CPU giải quyết vấn đề quyết định quá trình nào trong hàng đợi sẵn sàng được cấp phát CPU. Trong phần này chúng ta mô tả nhiều giải thuật định thời CPU đang có.

Định thời đến trước được phục vụ trước

Giải thuật định thời biểu CPU đơn giản nhất là đến trước, được phục vụ trước (first-come, first-served-FCFS). Với cơ chế này, quá trình yêu cầu CPU trước được cấp phát CPU trước. Việc cài đặt chính sách FCFS được quản lý dễ dàng với hàng đợi FIFO. Khi một quá trình đi vào hàng đợi sẵn sàng, PCB của nó được liên kết tới đuôi của hàng đợi. Khi CPU rảnh, nó được cấp phát tới một quá trình tại đầu hàng đợi. Sau đó, quá trình đang chạy được lấy ra khỏi hàng đợi. Mã của giải thuật FCFS đơn giản để viết và hiểu.

Tuy nhiên, thời gian chờ đợi trung bình dưới chính sách FCFS thường là dài. Xét tập hợp các quá trình sau đến tại thời điểm 0, với chiều dài thời gian chu kỳ CPU được cho theo mini giây.

Quá trình	Thời gian xử lý

P1	24
P2	3
P3	3

Nếu các quá trình đến theo thứ tự P1, P2, P3 và được phục vụ theo thứ tự FCFS, chúng ta nhận được kết quả được hiển thị trong lưu đồ Gantt như sau:

24	27	30

Thời gian chờ là 0 mili giây cho quá trình P1, 24 mili giây cho quá trình P2 và 27 mili giây cho quá trình P3. Do đó, thời gian chờ đợi trung bình là (0+24+27)/3=17 mili giây. Tuy nhiên, nếu các quá trình đến theo thứ tự P2, P3, P1 thì các kết quả được hiển thị trong lưu đồ Gannt như sau:

0 3	6	30

Thời gian chờ đợi trung bình bây giờ là (6+0+3)/3=3 mili giây. Việc cắt giảm này là quan trọng. Do đó, thời gian chờ đợi trung bình dưới chính sách FCFS thường không là tối thiểu và có sự thay đổi rất quan trọng nếu các thời gian CPU dành cho các quá trình khác nhau rất lớn.

Ngoài ra, xét năng lực của định thời FCFS trong trường hợp động. Giả sử chúng ta có một quá trình hướng xử lý (CPU-bound) và nhiều quá trình hướng nhập/xuất (I/O bound). Khi các quá trình đưa đến quanh hệ thống, ngữ cảnh sau có thể xảy ra. Quá trình hướng xử lý sẽ nhận CPU và giữ nó. Trong suốt thời gian này, tất cả quá trình khác sẽ kết thúc việc nhập/xuất của nó và chuyển vào hàng đợi sẵn sàng, các thiết bị nhập/xuất ở trạng thái rảnh. Cuối cùng, quá trình hướng xử lý kết thúc chu kỳ CPU của nó và chuyển tới thiết bị nhập/xuất. Tất cả các quá trình hướng xử lý có chu kỳ CPU rất ngắn sẽ nhanh chóng thực thi và di chuyển trở về hàng đợi nhập/xuất. Tại thời điểm này CPU ở trạng thái rảnh. Sau đó, quá trình hướng xử lý sẽ di chuyển trở lại hàng đợi sẵn sàng và được cấp CPU. Một lần nữa, tất cả quá trình hướng nhập/xuất kết thúc việc chờ trong hàng đợi sẵn sàng cho đến khi quá trình hướng xử lý được thực hiện. Có một tác dụng phụ (convoy effect) khi tất cả các quá trình khác chờ một quá trình lớn trả lại CPU. Tác dụng phụ này dẫn đến việc sử dụng thiết bị và CPU thấp hơn nếu các quá trình ngắn hơn được cấp trước.

Giải thuật FCSF là giải thuật định thời không trưng dụng CPU. Một khi CPU được cấp phát tới một quá trình, quá trình đó giữ CPU cho tới khi nó giải phóng CPU bằng cách kết thúc hay yêu cầu nhập/xuất. Giải thuật FCFS đặc biệt không phù hợp đối với hệ thống chia sẻ thời gian, ở đó mỗi người dùng nhận được sự chia sẻ CPU với những khoảng thời gian đều nhau.

Định thời biểu công việc ngắn nhất trước

Một tiếp cận khác đối với việc định thời CPU là giải thuật định thời công việc ngắn nhất trước (shortest-job-first-SJF). Giải thuật này gán tới mỗi quá trình chiều dài của chu kỳ CPU tiếp theo cho quá trình sau đó. Khi CPU sẵn dùng, nó được gán tới quá trình có chu kỳ CPU kế tiếp ngắn nhất. Nếu hai quá trình có cùng chiều dài chu kỳ CPU kế tiếp, định thời FCFS được dùng. Chú ý rằng thuật ngữ phù hợp hơn là chu kỳ CPU kế tiếp ngắn nhất (shortest next CPU burst) vì định thời được thực hiện bằng cách xem xét chiều dài của chu kỳ CPU kế tiếp của quá trình hơn là toàn bộ chiều dài của nó. Chúng ta dùng thuật ngữ SJF vì hầu hết mọi người và mọi sách tham khảo tới nguyên lý của loại định thời biểu này như SJF.

Thí dụ, xét tập hợp các quá trình sau, với chiều dài của thời gian chu kỳ CPU được tính bằng mili giây:

Quá trình	Thời gian xử lý
P1	6
P2	8
P3	7
P4	3

Dùng định thời SJF, chúng ta định thời biểu cho các quá trình này theo lưu đồ Gannt như sau:

0		3						9							16					

Thời gian chờ đợi là 3 mili giây cho quá trình P1, 16 mili giây cho quá trình P2, 9 mili giây cho quá trình P3, và 0 mili giây cho quá trình P4. Do đó, thời gian chờ đợi trung bình là (3+16+9+0)/4 = 7 mili giây. Nếu chúng ta dùng cơ chế định thời FCFS thì thời gian chờ đợi trung bình là 10.23 mili giây.

Giải thuật SJF có thể là tối ưu, trong đó nó cho thời gian chờ đợi trung bình nhỏ nhất cho các quá trình được cho. Bằng cách chuyển một quá trình ngắn trước một quá trình dài thì thời gian chờ đợi của quá trình ngắn giảm hơn so với việc tăng thời gian chờ đợi của quá trình dài. Do đó, thời gian chờ đợi trung bình giảm.

Khó khăn thật sự với giải thuật SJF là làm thế nào để biết chiều dài của yêu cầu CPU tiếp theo. Đối với định thời dài trong hệ thống bó, chúng ta có thể dùng chiều dài như giới hạn thời gian xử lý mà người dùng xác định khi gởi công việc. Do đó, người dùng được cơ động để ước lượng chính xác giới hạn thời gian xử lý vì giá trị thấp hơn có nghĩa là đáp ứng nhanh hơn. Định thời SJF được dùng thường xuyên trong định thời dài.

Mặc dù SJF là tối ưu nhưng nó không thể được cài đặt tại cấp định thời CPU ngắn vì không có cách nào để biết chiều dài chu kỳ CPU tiếp theo. Một tiếp cận là khác gần đúng định thời SJF được thực hiện. Chúng ta có thể không biết chiều dài của chu kỳ CPU kế tiếp nhưng chúng ta có đoán giá trị của nó. Chúng ta mong muốn rằng chu kỳ CPU kế tiếp sẽ tương tự chiều dài những chu kỳ CPU trước đó. Do đó, bằng cách tính toán mức xấp xỉ chiều dài của chu kỳ CPU kế tiếp, chúng ta chọn một quá trình với chu kỳ CPU được đoán là ngắn nhất.

Chu kỳ CPU kế tiếp thường được đoán như trung bình số mũ của chiều dài các chu kỳ CPU trước đó. Gọi tn là chiều dài của chu kỳ CPU thứ n và gọi Tn+1 giá trị được đoán cho chu kỳ CPU kế tiếp. Thì đối với α, với 0 ≤ α ≤ 1, định nghĩa

Tn+1 = α tn + (1- α) Tn

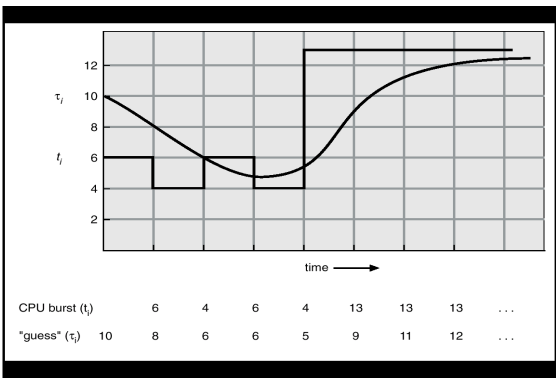
Công thức này định nghĩa một giá trị trung bình số mũ. Giá trị của tn chứa thông tin mới nhất; Tn lưu lịch sử quá khứ. Tham số α điều khiển trọng số liên quan giữa lịch sử quá khứ và lịch sử gần đây trong việc đoán. Nếu α=0 thì Tn+1=Tn và lịch sử gần đây không có ảnh hưởng (điều kiện hiện hành được đảm bảo là ngắn); nếu α =1 thì Tn+1=tn và chỉ chu kỳ CPU gần nhất có ảnh hưởng (lịch sử được đảm bảo là cũ và không phù hợp). Thông dụng hơn, α=1/2 thì lịch sử gần đây và lịch sử quá khứ có trọng số tương đương. Giá trị khởi đầu T0 có thể được định nghĩa như một hằng số hay như toàn bộ giá trị trung bình hệ thống. Hình IV.2 dưới đây hiển thị giá trị trung bình dạng mũ với α=1/2 và T0=10.

Để hiểu hành vi của giá trị trung bình dạng mũ, chúng ta có thể mở rộng công thức cho Tn+1 bằng cách thay thế Tn để tìm

Tn+1=α tn+(1-α) α tn-1+…+(1-α)j α tn-j+…+(1-α)n - 1T0

Vì cả hai α và (1- α) là nhỏ hơn hay bằng 1, mỗi số hạng tiếp theo có trọng số nhỏ hơn số hạng trước đó.

Giải thuật SJF có thể trưng dụng hoặc không trưng dụng CPU. Chọn lựa này phát sinh khi một quá trình mới đến tại hàng đợi sẵn sàng trong khi một quá trình trước đó đang thực thi. Một quá trình mới có thể có chu kỳ CPU tiêp theo ngắn hơn chu kỳ CPU được để lại của quá trình thực thi hiện tại. Giải thuật SJF trưng dụng sẽ trưng dungj CPU của quá trình đang thực thi hiện tại, trong khi giải thuật SJF không trưng dụng sẽ cho phép quá trình đang thực thi kết thúc chu kỳ CPU của nó. Định thời SJF trưng dụng còn được gọi là định thời thời gian còn lại ngắn nhất trước (shortest-remaining-time-first).



Hình IV-2 Đoán chiều dài của chu kỳ CPU kế tiếp

Thí dụ, xét 4 quá trình sau với chiều dài của thời gian chu kỳ CPU được cho tính bằng mili giây

Quá trình	Thời gian đến	Thời gian xử lý
P1	0	8

P2	1	4
P3	2	9
P4	3	5

Nếu các quá trình đi vào hàng đợi sẵn sàng tại những thời điểm và cần thời gian xử lý được hiển thị trong bảng trên thì thời biểu SJF trưng dụng được mô tả trong lưu đồ Gannt như sau:

P1	P2	P3		
0	1 5	10	17	26

Quá trình P1 được bắt đầu tại thời điểm 0, vì nó là quá trình duy nhất trong hàng đợi. Quá trình P2 đến tại thời điểm 1. Thời gian còn lại cho P1 (7 mili giây) là lớn hơn thời gian được yêu cầu bởi quá trình P2 (4 mili giây) vì thế quá trình P1 bị trưng dụng CPU và quá trình P2 được định thời biểu. Thời gian chờ đợi trung bình cho thí dụ này là: ((10-1) + (1-1) + (17-2) + (5-3))/4 = 6.5 mili giây. Định thời SJF không trưng dụng cho kết quả thời gian chờ đợi trung bình là 7.75 mili giây.

Định thời theo độ ưu tiên

Giải thuật SJF là trường hợp đặc biệt của giải thuật định thời theo độ ưu tiên (priority-scheduling algorithm). Độ ưu tiên được gán với mỗi quá trình và CPU được cấp phát tới quá trình với độ ưu tiên cao nhất. Quá trình có độ ưu tiên bằng nhau được định thời trong thứ tự FCFS.

Giải thuật SJF là giải thuật ưu tiên đơn giản ở đó độ ưu tiên p là nghịch đảo với chu kỳ CPU được đoán tiếp theo. Chu kỳ CPU lớn hơn có độ ưu tiên thấp hơn và ngược lại.

Bây giờ chúng ta thảo luận định thời có độ ưu tiên cao và thấp. Các độ ưu tiên thường nằm trong dãy số cố định, chẳng hạn 0 tới 7 hay 0 tới 4,095. Tuy nhiên, không có sự thoả thuận chung về 0 là độ ưu tiên thấp nhất hay cao nhất. Một vài hệ thống dùng số thấp để biểu diễn độ ưu tiên thấp; ngược lại các hệ thống khác dùng các số thấp cho độ ưu tiên cao. Sự khác nhau này có thể dẫn đến sự lẫn lộn. Trong giáo trình này chúng ta dùng các số thấp để biểu diễn độ ưu tiên cao.

Thí dụ, xét tập hợp quá trình sau đến tại thời điểm 0 theo thứ tự P1, P2,…, P5 với chiều dài thời gian chu kỳ CPU được tính bằng mili giây:

Quá trình	Thời gian xử lý	Độ ưu tiên
P1	10	3
P2	1	1

P3	2	4
P4	1	5
P5	5	2

Sử dụng định thời theo độ ưu tiên, chúng ta sẽ định thời các quá trình này theo lưu đồ Gannt như sau:

P2	P5	P1	P3	P4
0	1 6	16	18	19

Thời gian chờ đợi trung bình là 8.2 mili giây.

Độ ưu tiên có thể được định nghĩa bên trong hay bên ngoài. Độ ưu tiên được định nghĩa bên trong thường dùng định lượng hoặc nhiều định lượng có thể đo để tính toán độ ưu tiên của một quá trình. Thí dụ, các giới hạn thời gian, các yêu cầu bộ nhớ, số lượng tập tin đang mở và tỉ lệ của chu kỳ nhập/xuất trung bình với tỉ lệ của chu kỳ CPU trung bình. Các độ ưu tiên bên ngoài được thiết lập bởi các tiêu chuẩn bên ngoài đối với hệ điều hành như sự quan trọng của quá trình, loại và lượng chi phí đang được trả cho việc dùng máy tính, văn phòng hỗ trợ công việc, ..

Định thời biểu theo độ ưu tiên có thể trưng dụng hoặc không trưng dụng CPU. Khi một quá trình đến hàng đợi sẵn sàng, độ ưu tiên của nó được so sánh với độ ưu tiên của quá trình hiện đang chạy. Giải thuật định thời theo độ ưu tiên trưng dụng sẽ chiếm CPU nếu độ ưu tiên của quá trình mới đến cao hơn độ ưu tiên của quá trình đang thực thi. Giải thuật định thời theo độ ưu tiên không trưng dụng sẽ đơn giản đặt quá trình mới tại đầu hàng đợi sẵn sàng.

Vấn đề chính với giải thuật định thời theo độ ưu tiên là nghẽn không hạn định (indefinite blocking) hay đói CPU (starvation). Một quá trình sẵn sàng chạy nhưng thiếu CPU có thể xem như bị nghẽn-chờ đợi CPU. Giải thuật định thời theo độ ưu tiên có thể để lại nhiều quá trình có độ ưu tiên thấp chờ CPU không hạn định. Trong một hệ thống máy tính tải cao, dòng đều đặn các quá trình có độ ưu tiên cao hơn có thể ngăn chặn việc nhận CPU của quá trình có độ ưu tiên thấp.. Thông thường, một trong hai trường hợp xảy ra. Cuối cùng, một quá trình sẽ được chạy (lúc 2 a.m chủ nhật là thời điểm cuối cùng hệ thống nạp các quá trình nhẹ), hay cuối cùng hệ thống máy tính sẽ đổ vỡ và mất tất cả các quá trình có độ ưu tiên thấp chưa được kết thúc.

Một giải pháp cho vấn đề nghẽn không hạn định này là sự hoá già (aging). Hóa già là kỹ thuật tăng dần độ ưu tiên của quá trình chờ trong hệ thống một thời gian dài. Thí dụ, nếu các độ ưu tiên nằm trong dãy từ 127 (thấp) đến 0 (cao), chúng ta giảm độ ưu tiên của quá trình đang chờ xuống 1 mỗi 15 phút. Cuối cùng, thậm chí một quá trình với độ ưu tiên khởi đầu 127 sẽ đạt độ ưu tiên cao nhất trong hệ thống và sẽ được thực thi. Thật vậy, một quá trình sẽ mất không quá 32 giờ để đạt được độ ưu tiên từ 127 tới 0.

Định thời luân phiên

Giải thuật định thời luân phiên (round-robin scheduling algorithm-RR) được thiết kế đặc biệt cho hệ thống chia sẻ thời gian. Tương tự như định thời FCFS nhưng sự trưng dụng CPU được thêm vào để chuyển CPU giữa các quá trình. Đơn vị thời gian nhỏ được gọi là định mức thời gian (time quantum) hay phần thời gian (time slice) được định nghĩa. Định mức thời gian thường từ 10 đến 100 mili giây. Hàng đợi sẵn sàng được xem như một

hàng đợi vòng. Bộ định thời CPU di chuyển vòng quanh hàng đợi sẵn sàng, cấp phát CPU tới mỗi quá trình có khoảng thời gian tối đa bằng một định mức thời gian.

Để cài đặt định thời RR, chúng ta quản lý hàng đợi sẵn sàng như một hàng đợi FIFO của các quá trình. Các quá trình mới được thêm vào đuôi hàng đợi. Bộ định thời CPU chọn quá trình đầu tiên từ hàng đợi sẵn sàng, đặt bộ đếm thời gian để ngắt sau 1 định mức thời gian và gởi tới quá trình.

Sau đó, một trong hai trường hợp sẽ xảy ra. Quá trình có 1 chu kỳ CPU ít hơn 1 định mức thời gian. Trong trường hợp này, quá trình sẽ tự giải phóng. Sau đó, bộ định thời biểu sẽ xử lý quá trình tiếp theo trong hàng đợi sẵn sàng. Ngược lại, nếu chu kỳ CPU của quá trình đang chạy dài hơn 1 định mức thời gian thì độ đếm thời gian sẽ báo và gây ra một ngắt tới hệ điều hành. Chuyển đổi ngữ cảnh sẽ được thực thi và quá trình được đặt trở lại tại đuôi của hàng đợi sẵn sàng. Sau đó, bộ định thời biểu CPU sẽ chọn quá trình tiếp theo trong hàng đợi sẵn sàng.

Tuy nhiên, thời gian chờ đợi trung bình dưới chính sách RR thường là quá dài. Xét một tập hợp các quá trình đến tại thời điểm 0 với chiều dài thời gian CPU-burst được tính bằng mili giây:

Quá trình	Thời gian xử lý
P1	24
P2	3
P3	3

Nếu sử dụng định mức thời gian là 4 mili giây thì quá trình P1 nhận 4 mili giây đầu tiên. Vì nó yêu cầu 20 mili giây còn lại nên nó bị trưng dụng CPU sau định mức thời gian đầu tiên và CPU được cấp tới quá trình tiếp theo trong hàng đợi, quá trình P2. Vì P2 không cần tới 4 mili giây nên nó kết thúc trước khi định mức thời gian của nó hết hạn. Sau đó, CPU được cho tới quá trình kế tiếp, quá trình P3. Một khi mỗi quá trình nhận 1 định mức thời gian thì CPU trả về quá trình P1 cho định mức thời gian tiếp theo. Thời biểu RR là:

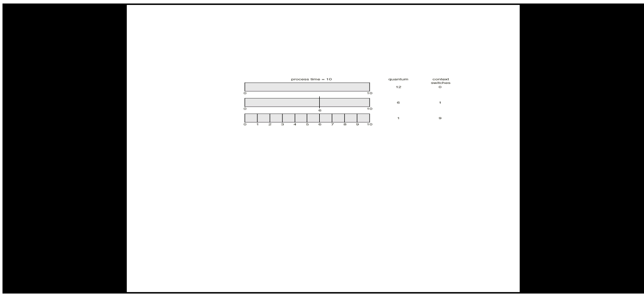
0 4	7	10	14	18	22	26	30

Thời gian chờ đợi trung bình là 17/3=5.66 mili giây.

Trong giải thuật RR, không quá trình nào được cấp phát CPU cho nhiều hơn 1 định mức thời gian trong một hàng. Nếu chu kỳ CPU của quá trình vượt quá 1 định mức thời gian thì quá trình đó bị trưng dụng CPU và nó được đặt trở lại hàng đợi sẵn sàng. Giải thuật RR là giải thuật trưng dụng CPU.

Nếu có n quá trình trong hàng đợi sẵn sàng và định mức thời gian là q thì mỗi quá trình nhận 1/n thời gian CPU trong các phần, nhiều nhất q đơn vị thời gian. Mỗi quá trình sẽ chờ không dài hơn (n-1)x q đơn vị thời gian cho tới khi định mức thời gian tiếp theo của nó. Thí dụ, nếu có 5 quá trình với định mức thời gian 20 mili giây thì mỗi quá trình sẽ nhận 20 mili giây sau mỗi 100 mili giây.

Năng lực của giải thuật RR phụ thuộc nhiều vào kích thước của định mức thời gian. Nếu định mức thời gian rất lớn (lượng vô hạn) thì chính sách RR tương tự như chính sách FCFS. Nếu định mức thời gian là rất nhỏ (1 mili giây) thì tiếp cận RR được gọi là chia sẻ bộ xử lý (processor sharing) và xuất hiện (trong lý thuyết) tới người dùng như thể mỗi quá trình trong n quá trình có bộ xử lý riêng của chính nó chạy tại 1/n tốc độ của bộ xử lý thật.

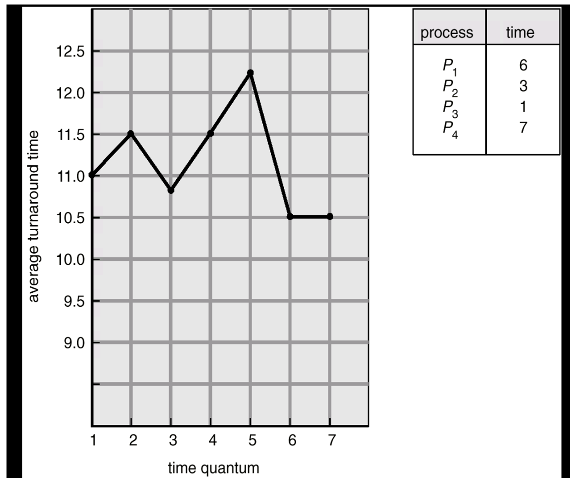


Hình IV-3 Hiển thị một định mức thời gian nhỏ hơn tăng chuyển đổi ngữ cảnh như thế nào

Tuy nhiên, trong phần mềm chúng ta cũng cần xem xét hiệu quả của việc chuyển đổi ngữ cảnh trên năng lực của việc định thời RR. Chúng ta giả sử rằng chỉ có 1 quá trình với 10 đơn vị thời gian. Nếu một định mức là 12 đơn vị thời gian thì quá trình kết thúc ít hơn 1 định mức thời gian, với không có chi phí nào khác. Tuy nhiên, nếu định mức là 6 đơn vị thời gian thì quá trình cần 2 định mức thời gian, dẫn đến 1 chuyển đổi ngữ cảnh. Nếu định mức thời gian là 1 đơn vị thời gian thì 9 chuyển đổi ngữ cảnh sẽ xảy ra, việc thực thi của quá trình bị chậm như được hiển thị trong hình IV.3 .

Do đó chúng ta mong muốn định mức thời gian lớn đối với thời gian chuyển ngữ cảnh. Nếu thời gian chuyển ngữ cảnh chiếm 10% định mức thời gian thì khoảng 10% thời gian CPU sẽ được dùng cho việc chuyển ngữ cảnh.

Thời gian hoàn thành cũng phụ thuộc kích thước của định mức thời gian. Chúng ta có thể thấy trong hình IV.4, thời gian hoàn thành trung bình của tập hợp các quá trình không cần cải tiến khi kích thước định mức thời gian tăng. Thông thường, thời gian hoàn thành trung bình có thể được cải tiến nếu hầu hết quá trình kết thúc chu kỳ CPU kế tiếp của chúng trong một định mức thời gian. Thí dụ, cho 3 quá trình có 10 đơn vị thời gian cho mỗi quá trình và định mức thời gian là 1 đơn vị thời gian, thì thời gian hoàn thành trung bình là 29. Tuy nhiên, nếu định mức thời gian là 10 thì thời gian hoàn thành trung bình giảm tới 20. Nếu thời gian chuyển ngữ cảnh được thêm vào thì thời gian hoàn thành trung bình gia tăng đối với định mức thời gian nhỏ hơn vì các chuyển đổi ngữ cảnh thêm nữa sẽ được yêu cầu.



Hình IV-4 Hiển thị cách thời gian hoàn thành biến đổi theo định mức thời gian

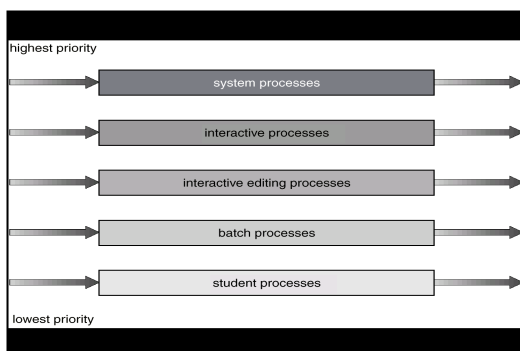
Ngoài ra, nếu định mức thời gian quá lớn thì người thiết kế việc định thời RR bao gồm chính sách FCFS. Qui tắc là định mức thời gian nên dài hơn 80% chu kỳ CPU.

Định thời biểu với hàng đợi nhiều cấp

Một loại giải thuật định thời khác được tạo ra cho những trường hợp mà trong đó các quá trình được phân lớp thành các nhóm khác nhau. Thí dụ: việc phân chia thông thường được thực hiện giữa các quá trình chạy ở chế độ giao tiếp (foreground hay interactive) và các quá trình chạy ở chế độ nền hay dạng bó (background hay batch). Hai loại quá trình này có yêu cầu đáp ứng thời gian khác nhau và vì thế có yêu cầu về định thời biểu khác nhau. Ngoài ra, các quá trình chạy ở chế độ giao tiếp có độ ưu tiên (hay được định nghĩa bên ngoài) cao hơn các quá trình chạy ở chế độ nền.

Một giải thuật định thời hàng đợi nhiều cấp (multilevel queue-scheduling algorithm) chia hàng đợi thành nhiều hàng đợi riêng rẻ (hình IV.5). Các quá trình được gán vĩnh viễn tới một hàng đợi, thường dựa trên thuộc tính của quá trình như kích thước bộ nhớ, độ ưu tiên quá trình hay loại quá trình. Mỗi hàng đợi có giải thuật định thời của chính nó. Thí dụ: các hàng đợi riêng rẻ có thể được dùng cho các quá trình ở chế độ nền và chế độ giao tiếp. Hàng đợi ở chế độ giao tiếp có thể được định thời bởi giải thuật RR trong khi hàng đợi ở chế độ nền được định thời bởi giải thuật FCFS.

Ngoài ra, phải có việc định thời giữa các hàng đợi, mà thường được cài đặt như định thời trưng dụng với độ ưu tiên cố định. Thí dụ, hàng đợi ở chế độ giao tiếp có độ ưu tiên tuyệt đối hơn hàng đợi ở chế độ nền.



Hình IV-5 Định thời hàng đợi nhiều mức

Chúng ta xét một thí dụ của giải thuật hàng đợi nhiều mức với năm hàng đợi:

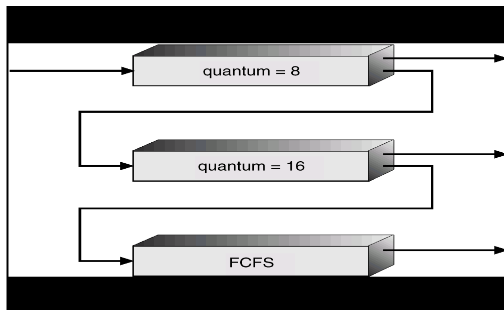
1. Các quá trình hệ thống
2. Các quá trình giao tiếp
3. Các quá trình soạn thảo giao tiếp
4. Các quá trình bó
5. Các quá trình sinh viên

Mỗi hàng đợi có độ ưu tiên tuyệt đối hơn hàng đợi có độ ưu tiên thấp hơn. Thí dụ: không có quá trình nào trong hàng đợi bó có thể chạy trừ khi hàng đợi cho các quá trình hệ thống, các quá trình giao tiếp và các quá trình soạn thảo giao tiếp đều rỗng. Nếu một quá trình soạn thảo giao tiếp được đưa vào hàng đợi sẵn sàng trong khi một quá trình bó đang chạy thì quá trình bó bị trưng dụng CPU. Solaris 2 dùng dạng giải thuật này.

Một khả năng khác là phần (slice) thời gian giữa hai hàng đợi. Mỗi hàng đợi nhận một phần thời gian CPU xác định, sau đó nó có thể định thời giữa các quá trình khác nhau trong hàng đợi của nó. Thí dụ, trong hàng đợi giao tiếp-nền, hàng đợi giao tiếp được cho 80% thời gian của CPU cho giải thuật RR giữa các quá trình của nó, ngược lại hàng đợi nền nhận 20% thời gian CPU cho các quá trình của nó theo cách FCFS.

Định thời hàng đợi phản hồi đa cấp

Thông thường, trong giải thuật hàng đợi đa cấp, các quá trình được gán vĩnh viễn tới hàng đợi khi được đưa vào hệ thống. Các quá trình không di chuyển giữa các hàng đợi. Nếu có các hàng đợi riêng cho các quá trình giao tiếp và các quá trình nền thì các quá trình không di chuyển từ một hàng đợi này tới hàng đợi khác vì các quá trình không thay đổi tính tự nhiên giữa giao tiếp và nền. Cách tổ chức có ích vì chi phí định thời thấp nhưng thiếu linh động và có thể dẫn đến tình trạng "đói CPU".



Hình IV-6 Các hàng đợi phản hồi nhiều cấp

Tuy nhiên, định thời hàng đợi phản hồi đa cấp (multilevel feedback queue scheduling) cho phép một quá trình di chuyển giữa các hàng đợi. Ý tưởng là tách riêng các quá trình với các đặc điểm chu kỳ CPU khác nhau. Nếu một quá trình dùng quá nhiều thời gian CPU thì nó sẽ được di chuyển tới hàng đợi có độ ưu tiên thấp. Cơ chế này để lại các quá trình hướng nhập/xuất và các quá trình giao tiếp trong các hàng đợi có độ ưu tiên cao hơn. Tương tự, một quá trình chờ quá lâu trong hàng đợi có độ ưu tiên thấp hơn có thể được di chuyển tới hàng đợi có độ ưu tiên cao hơn. Đây là hình thức của sự hóa già nhằm ngăn chặn sự đói CPU.

Thí dụ, xét một bộ định thời hàng đợi phản hồi nhiều cấp với ba hàng đợi được đánh số từ 0 tới 2 (như hình IV.6). Bộ định thời trước tiên thực thi tất cả quá trình chứa trong hàng đợi 0. Chỉ khi hàng đợi 0 rỗng nó sẽ thực thi các quá trình trong hàng đợi 1. Tương tự, các quá trình trong hàng đợi 2 sẽ được thực thi chỉ nếu hàng đợi 0

và 1 rỗng. Một quá trình đến hàng đợi 1 sẽ ưu tiên hơn quá trình đến hàng đợi 2. Tương tự, một quá trình đến hàng đợi 0 sẽ ưu tiên hơn một quá trình vào hàng đợi 1.

Một quá trình đưa vào hàng đợi sẵn sàng được đặt trong hàng đợi 0. Một quá trình trong hàng đợi 0 được cho một định mức thời gian là 8 mili giây. Nếu nó không kết thúc trong thời gian này thì nó sẽ di chuyển vào đuôi của hàng đợi 1. Nếu hàng đợi 0 rỗng thì quá trình tại đầu của hàng đợi 1 được cho định mức thời gian là 16 mili giây. Nếu nó không hoàn thành thì nó bị chiếm CPU và được đặt vào hàng đợi 2. Các quá trình trong hàng đợi 2 được chạy trên cơ sở FCFS chỉ khi hàng đợi 0 và 1 rỗng.

Giải thuật định thời này cho độ ưu tiên cao nhất tới bất cứ quá trình nào với chu kỳ CPU 8 mili giây hay ít hơn. Một quá trình như thế sẽ nhanh chóng nhận CPU, kết thúc chu kỳ CPU của nó và bỏ đi chu kỳ I/O kế tiếp của nó. Các quá trình cần hơn 8 mili giây nhưng ít hơn 24 mili giây được phục vụ nhanh chóng mặc dù với độ ưu tiên thấp hơn các quá trình ngắn hơn. Các quá trình dài tự động rơi xuống hàng đợi 2 và được phục vụ trong thứ tự FCFS với bất cứ chu kỳ CPU còn lại từ hàng đợi 0 và 1.

Nói chung, một bộ định thời hàng đợi phản hồi nhiều cấp được định nghĩa bởi các tham số sau:

- Số lượng hàng đợi
- Giải thuật định thời cho mỗi hàng đợi
- Phương pháp được dùng để xác định khi nâng cấp một quá trình tới hàng đợi có độ ưu tiên cao hơn.
- Phương pháp được dùng để xác định khi nào chuyển một quá trình tới hàng đợi có độ ưu tiên thấp hơn.
- Phương pháp được dùng để xác định hàng đợi nào một quá trình sẽ đi vào và khi nào quá trình đó cần phục vụ.

Định nghĩa bộ định thời biểu dùng hàng đợi phản hồi nhiều cấp trở thành giải thuật định thời CPU phổ biến nhất. Bộ định thời này có thể được cấu hình để thích hợp với hệ thống xác định. Tuy nhiên, bộ định thời này cũng yêu cầu một vài phương tiện chọn lựa giá trị cho tất cả tham số để định nghĩa bộ định thời biểu tốt nhất. Mặc dù một hàng đợi phản hồi nhiều cấp là cơ chế phổ biến nhất nhưng nó cũng là cơ chế phức tạp nhất.

Định thời biểu đa bộ xử lý

Phần trên thảo luận chúng ta tập trung vào những vấn đề định thời biểu CPU trong một hệ thống với một bộ vi xử lý đơn. Nếu có nhiều CPU, vấn đề định thời tương ứng sẽ phức tạp hơn. Nhiều khả năng đã được thử nghiệm và như chúng ta đã thấy với định thời CPU đơn bộ xử lý, không có giải pháp tốt nhất. Trong phần sau đây, chúng ta sẽ thảo luận vắn tắt một số vấn đề tập trung về định thời biểu đa bộ xử lý. Chúng ta tập trung vào những hệ thống mà các bộ xử lý của nó được xác định (hay đồng nhất) trong thuật ngữ chức năng của chúng; bất cứ bộ xử lý nào sẵn có thì có thể được dùng để chạy bất quá trình nào trong hàng đợi. Chúng ta cũng cho rằng truy xuất bộ nhớ là đồng nhất (uniform memory access-UMA). Chỉ những chương trình được biên dịch đối với tập hợp chỉ thị của bộ xử lý được cho mới có thể được chạy trên chính bộ xử lý đó.

Ngay cả trong một bộ đa xử lý đồng nhất đôi khi có một số giới hạn cho việc định thời biểu. Xét một hệ thống với một thiết bị nhập/xuất được gán tới một đường bus riêng của một bộ xử lý. Các quá trình muốn dùng thiết bị đó phải được định thời biểu để chạy trên bộ xử lý đó, ngược lại thiết bị đó là không sẵn dùng.

Nếu nhiều bộ xử lý xác định sẵn dùng thì chia sẻ tải có thể xảy ra. Nó có thể cung cấp một hàng đợi riêng cho mỗi bộ xử lý. Tuy nhiên, trong trường hợp này, một bộ xử lý có thể rảnh với hàng đợi rỗng, trong khi bộ xử lý khác rất bận. Để ngăn chặn trường hợp này, chúng ta dùng một hàng đợi sẵn sàng chung. Tất cả quá trình đi vào một hàng đợi và được định thời biểu trên bất cứ bộ xử lý sẵn dùng nào.

Trong một cơ chế như thế, một trong hai tiếp cận định thời biểu có thể được dùng. Trong tiếp cận thứ nhất, mỗi bộ xử lý định thời chính nó. Mỗi bộ xử lý xem xét hàng đợi sẵn sàng chung và chọn một quá trình để thực thi. Nếu chúng ta có nhiều bộ xử lý cố gắng truy xuất và cập nhật một cấu trúc dữ liệu chung thì mỗi bộ xử lý phải được lập trình rất cẩn thận. Chúng ta phải đảm bảo rằng hai bộ xử lý không chọn cùng quá trình và quá trình đó không bị mất từ hàng đợi. Tiếp cận thứ hai tránh vấn đề này bằng cách đề cử một bộ xử lý như bộ định thời cho các quá trình khác, do đó tạo ra cấu trúc chủ-tớ (master-slave).

Một vài hệ thống thực hiện cấu trúc này từng bước bằng cách tất cả quyết định định thời, xử lý nhập/xuất và các hoạt động hệ thống khác được quản lý bởi một bộ xử lý đơn-một server chủ. Các bộ xử lý khác chỉ thực thi mã người dùng. Đa xử lý không đối xứng (asymmetric multiprocessing) đơn giản hơn đa xử lý đối xứng (symmetric multiprocessing) vì chỉ một quá trình truy xuất các cấu trúc dữ liệu hệ thống, làm giảm đi yêu cầu chia sẻ dữ liệu. Tuy nhiên, nó cũng không hiệu quả. Các quá trình giới hạn nhập/xuất có thể gây thắt cổ chai (bottleneck) trên một CPU đang thực hiện tất cả các hoạt động. Điển hình, đa xử lý không đối xứng được cài đặt trước trong một hệ điều hành và sau đó được nâng cấp tới đa xử lý đối xứng khi hệ thống tiến triển.

Định thời thời gian thực

Trong chương đầu chúng ta đã tìm hiểu tổng quan về hệ điều hành thời thực và thảo luận tầm quan trọng của nó. Ở đây, chúng ta tiếp tục thảo luận bằng cách mô tả các điều kiện thuận lợi định thời cần để hỗ trợ tính toán thời thực trong hệ thống máy tính đa mục đích.

Tính toán thời thực được chia thành hai loại: hệ thống thời thực cứng (hardware real-time systems) được yêu cầu để hoàn thành một tác vụ tới hạn trong lượng thời gian được đảm bảo. Thông thường, một quá trình được đưa ra xem xét cùng với khai báo lượng thời gian nó cần để hoàn thành hay thực hiện nhập/xuất. Sau đó, bộ định thời biểu nhận được quá trình, đảm bảo rằng quá trình sẽ hoàn thành đúng giờ hay từ chối yêu cầu khi không thể. Điều này được gọi là đặt trước tài nguyên (resource reservation). Để đảm bảo như thế đòi hỏi bộ định thời biết chính xác bao lâu mỗi loại chức năng hệ điều hành mất để thực hiện và do đó mỗi thao tác phải được đảm bảo để mất lượng thời gian tối đa. Một đảm bảo như thế là không thể trong hệ thống với lưu trữ phụ và bộ nhớ ảo vì các hệ thống con này gây ra sự biến đổi không thể tránh hay không thể thấy trước trong lượng thời gian thực thi một quá trình xác định. Do đó, hệ thống thời thực cứng được hình thành từ nhiều phần mềm có mục đích đặc biệt chạy trên phần cứng tận hiến cho các quá trình tới hạn, và thiếu chức năng đầy đủ của các máy tính và các hệ điều hành hiện đại.

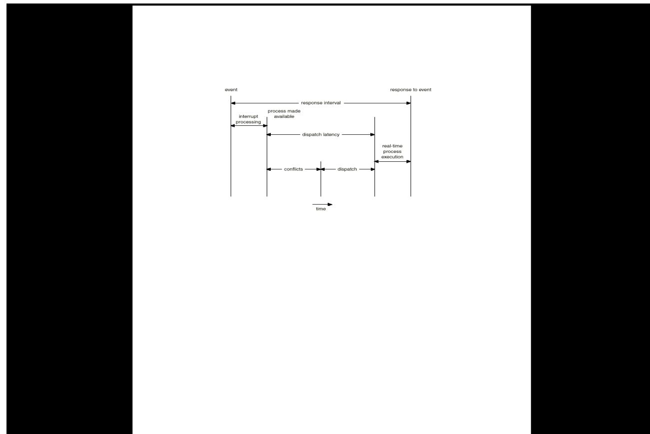
Tính toán thời gian thực mềm (soft real-time computing) ít nghiêm khắc hơn. Nó yêu cầu các quá trình tới hạn nhận độ ưu tiên cao hơn các quá trình khác. Mặc dù thêm chức năng thời thực mềm tới hệ chia sẻ thời gian có thể gây ra việc cấp phát tài nguyên không công bằng và có thể dẫn tới việc trì hoãn lâu hơn hay thậm chí đói tài nguyên đối với một số quá trình, nhưng nó ít có thể đạt được. Kết quả là hệ thống mục đích chung cũng có thể hỗ trợ đa phương tiện, đồ họa giao tiếp tốc độ cao, và nhiều tác vụ khác nhưng không hỗ trợ tính toán thời thực mềm.

Cài đặt chức năng thời thực mềm đòi hỏi thiết kế cẩn thận bộ định thời biểu và các khía cạnh liên quan của hệ điều hành. Trước tiên, hệ thống phải có định thời trưng dụng và các quá trình thời thực phải có độ ưu tiên cao nhất. Độ ưu tiên của các quá trình thời thực phải không giảm theo thời gian mặc dù độ ưu tiên của các quá trình không thời thực có thể giảm. Thứ hai, độ trễ của việc điều phối phải nhỏ. Một quá trình thời thực nhỏ hơn, nhanh hơn có thể bắt đầu thực thi một khi nó có thể chạy.

Quản trị các thuộc tính đã được xem xét ở trên là tương đối đơn giản. Thí dụ, chúng ta có thể không cho phép một quá trình hóa già trên các quá trình thời thực, do đó đảm bảo rằng độ ưu tiên của các quá trình không thay đổi. Tuy nhiên, đảm bảo thuộc tính sau đây phức tạp hơn. Vấn đề là nhiều hệ điều hành gồm hầu hết ấn bản của UNIX bị bắt buộc chờ lời gọi hệ thống hoàn thành hay nghẽn nhập/xuất xảy ra trước khi thực hiện chuyển ngữ cảnh. Độ trễ điều phối trong những hệ thống như thế có thể dài vì một số lời gọi hệ thống phức tạp và một vài thiết bị nhập/xuất chậm.

Để giữ độ trễ điều phối chậm, chúng ta cần cho phép các lời gọi hệ thống được trưng dụng. Có nhiều cách để đạt mục đích này. Cách thứ nhất là chèn các điểm trưng dụng (preemption points) trong những lời gọi hệ thống có khoảng thời gian dài, kiểm tra để thấy quá trình ưu tiên cao cần được thực thi hay không. Nếu đúng, thì chuyển ngữ cảnh xảy ra và khi quá trình có độ ưu tiên kết thúc, quá trình bị ngắt tiếp tục với lời gọi hệ thống. Các điểm trưng dụng chỉ có thể được đặt tại vị trí “an toàn” trong nhân- nơi mà những cấu trúc dữ liệu hiện tại không được cập nhật. Ngay cả với độ trễ điều phối trưng dụng có thể lớn vì chỉ một vài điểm trưng dụng có thể được thêm vào nhân trong thực tế.

Một phương pháp khác để giải quyết sự trưng dụng là làm toàn bộ nhân có thể trưng dụng. Để đảm bảo các hoạt động thực hiện đúng, tất cả cấu trúc dữ liệu nhân phải được bảo vệ thông qua việc sử dụng các cơ chế đồng bộ hóa. Với phương pháp này, nhân luôn có thể trưng dụng vì bất cứ dữ liệu nhân được cập nhật được bảo vệ từ việc sửa đổi bởi quá trình có độ ưu tiên cao. Đây là một phương pháp hiệu quả nhất trong việc sử dụng rộng rãi; nó được dùng trong Solaris 2.



Hình IV-7 Độ trễ gửi

Nhưng điều gì xảy ra nếu quá trình có độ ưu tiên cao cần đọc hay sửa đổi dữ liệu nhân hiện đang được truy xuất bởi quá trình khác có độ ưu tiên thấp hơn? Quá trình có độ ưu tiên cao đang chờ quá trình có độ ưu tiên thấp kết thúc. Trường hợp này được gọi là đảo ngược độ ưu tiên (prioprity inversion). Thật vậy, một chuỗi các quá trình đang truy xuất tài nguyên mà quá trình có độ ưu tiên cao cần. Vấn đề này có thể giải quyết bằng giao thức kế thừa độ ưu tiên (priority-inheritance protocol) trong đó tất cả quá trình này (các quá trình này truy xuất tài nguyên mà quá trình có độ ưu tiên cao cần) kế thừa độ ưu tiên cao cho đến khi chúng được thực hiện với tài nguyên trong câu hỏi. Khi chúng kết thúc, độ ưu tiên của chúng chuyển trở lại giá trị ban đầu của nó.

Trong hình IV.7, chúng ta hiển thị sự thay đổi của độ trễ điều phối. Giai đoạn xung đột (conflict phase) của độ trễ điều phối có hai thành phần:

1. Sự trưng dụng của bất cứ quá trình nào đang chạy trong nhân
2. Giải phóng tài nguyên các quá trình có độ ưu tiên thấp được yêu cầu bởi quá trình có độ ưu tiên cao

Thí dụ, trong Solaris 2 độ trễ điều phối với sự trưng dụng bị vô hiệu hóa khi vượt qua 100 mili giây. Tuy nhiên, độ trễ điều phối với sự trưng dụng được cho phép thường được giảm xuống tới 2 mili giây.

Đánh giá giải thuật

Chúng ta chọn một giải thuật định thời CPU cho một hệ thống xác định như thế nào? Có nhiều giải thuật định thời, mỗi giải thuật với các tham số của riêng nó. Do đó, chọn một giải thuật có thể là khó.

Vấn đề đầu tiên là định nghĩa các tiêu chuẩn được dùng trong việc chọn một giải thuật. Các tiêu chuẩn thường được định nghĩa trong thuật ngữ khả năng sử dụng CPU, thời gian đáp ứng hay thông lượng. Để chọn một giải thuật, trước hết chúng ta phải định nghĩa trọng số quan trọng của các thước đo này. Tiêu chuẩn của chúng ta có thể gồm các thước đo như:

- Khả năng sử dụng CPU tối đa dưới sự ràng buộc thời gian đáp ứng tối đa là 1 giây.
- Thông lượng tối đa như thời gian hoàn thành (trung bình) tỉ lệ tuyến tính với tổng số thời gian thực thi.

Một khi các tiêu chuẩn chọn lựa được định nghĩa, chúng ta muốn đánh giá các giải thuật khác nhau dưới sự xem xét. Chúng ta mô tả các phương pháp đánh giá khác nhau trong những phần dưới đây

Mô hình quyết định

Một loại quan trọng của phương pháp đánh giá được gọi là đánh giá phân tích (analytic evaluation). Đánh giá phân tích dùng giải thuật được cho và tải công việc hệ thống để tạo ra công thức hay số đánh giá năng lực của giải thuật cho tải công việc đó.

Một dạng đánh giá phân tích là mô hình xác định (deterministic modeling). Phương pháp này lấy tải công việc đặc biệt được xác định trước và định nghĩa năng lực của mỗi giải thuật cho tải công việc đó.

Thí dụ, giả sử rằng chúng ta có tải công việc được hiển thị trong bảng dưới. Tất cả 5 quá trình đến tại thời điểm 0 trong thứ tự được cho, với chiều dài của thời gian chu kỳ CPU được tính bằng mili giây.

Quá trình	Thời gian xử lý
P1	10
P2	29
P3	3
P4	7
P5	12

Xét giải thuật định thời FCFS, SJF và RR (định mức thời gian=10 mili giây) cho tập hợp quá trình này. Giải thuật nào sẽ cho thời gian chờ đợi trung bình tối thiểu?

Đối với giải thuật FCFS, chúng ta sẽ thực thi các quá trình này như sau:

P1	P2			P3	P4	P5
0 10			39	42	49	61

Thời gian chờ đợi là 0 mili giây cho quá trình P1, 32 mili giây cho quá trình P2, 39 giây cho quá trình P3, 42 giây cho quá trình P4 và 49 mili giây cho quá trình P5. Do đó, thời gian chờ đợi trung bình là (0 + 10 + 39 + 42 + 49)/5= 28 mili giây.

Với định thời không trưng dụng SJF, chúng ta thực thi các quá trình như sau:

P3	P4		P1	P5	P2		
0	3	10	20	32			61

Thời gian chờ đợi là 10 mili giây cho quá trình P1, 32 mili giây cho quá trình P2, 0 mili giây cho quá trình P3, 3 mili giây cho quá trình P4, và 20 giây cho quá trình P5. Do đó, thời gian chờ đợi trung bình là (10 + 32 + 0 + 3 + 20)/5= 13 mili giây.

Với giải thuật RR, chúng ta thực thi các quá trình như sau:

P1	P2	P3	P4	P5	P2	P5	P2
0 10	20	23	30	40	50	52	61

Thời gian chờ đợi là 0 mili giây cho quá trình P1, 32 mili giây cho quá trình P2, 20 mili giây cho quá trình P3, 23 mili giây cho quá trình P4, và 40 mili giây cho quá trình P5. Do đó, thời gian chờ đợi trung bình là (0 + 32 + 20 + 23 + 40)/5 = 23 mili giây.

Trong trường hợp này, chúng ta thấy rằng, chính sách SJF cho kết quả ít hơn ½ thời gian chờ đợi trung bình đạt được với giải thuật FCFS; giải thuật RR cho chúng ta giá trị trung gian.

Mô hình xác định là đơn giản và nhanh. Nó cho các con số chính xác, cho phép các giải thuật được so sánh với nhau. Tuy nhiên, nó đòi hỏi các số đầu vào chính xác và các trả lời của nó chỉ áp dụng cho những trường hợp đó. Việc dùng chủ yếu của mô hình xác định là mô tả giải thuật định thời và cung cấp các thí dụ. Trong các trường hợp, chúng ta đang chạy cùng các chương trình lặp đi lặp lại và có thể đo các yêu cầu xử lý của chương trình một cách chính xác, chúng ta có thể dùng mô hình xác định để chọn giải thuật định thời. Qua tập hợp các thí dụ, mô hình xác định có thể hiển thị khuynh hướng được phân tích và chứng minh riêng. Thí dụ, có thể chứng minh rằng đối với môi trường được mô tả (tất cả quá trình và thời gian của chúng sẵn dùng tại thời điểm 0), chính sách SJF sẽ luôn cho kết quả thời gian chờ đợi là nhỏ nhất.

Tuy nhiên, nhìn chung mô hình xác định quá cụ thể và yêu cầu tri thức quá chính xác để sử dụng nó một cách có ích.

Mô hình hàng đợi

Các quá trình được chạy trên nhiều hệ thống khác nhau từ ngày này sang ngày khác vì thế không có tập hợp quá trình tĩnh (và thời gian) để dùng cho mô hình xác định. Tuy nhiên, những gì có thể được xác định là sự phân bổ chu kỳ CPU và I/O. Sự phân bổ này có thể được đo và sau đó được tính xấp xỉ hay ước lượng đơn giản. Kết quả là một công thức toán mô tả xác suất của một chu kỳ CPU cụ thể. Thông thường, sự phân bổ này là hàm mũ và được mô tả bởi giá trị trung bình của nó. Tương tự, sự phân bổ thời gian khi các quá trình đến trong hệ thống-phân bổ thời gian đến-phải được cho.

Hệ thống máy tính được mô tả như một mạng các server. Mỗi server có một hàng đợi cho các quá trình. CPU là một server với hàng đợi sẵn sàng của nó, như là một hệ thống nhập/xuất với các hàng đợi thiết bị. Biết tốc độ đến và tốc độ phục vụ, chúng ta có thể tính khả năng sử dụng, chiều dài hàng đợi trung bình, thời gian chờ trung bình,..Lĩnh vực nghiên cứu này được gọi là phân tích mạng hàng đợi (queueing-network analysis).

Thí dụ, gọi n là chiều dài hàng đợi trung bình (ngoại trừ các quá trình đang được phục vụ), gọi W là thời gian chờ đợi trung bình trong hàng đợi và λ là tốc độ đến trung bình cho các quá trình mới trong hàng đợi (chẳng hạn 3 quá trình trên giây). Sau đó, chúng ta mong đợi trong suốt thời gian W một quá trình chờ, λ x W các quá trình mới sẽ đến trong hàng đợi. Nếu hệ thống ở trong trạng thái đều đặn thì số lượng quá trình rời hàng đợi phải bằng số lượng quá trình đến. Do đó,

$n = \lambda \times W$

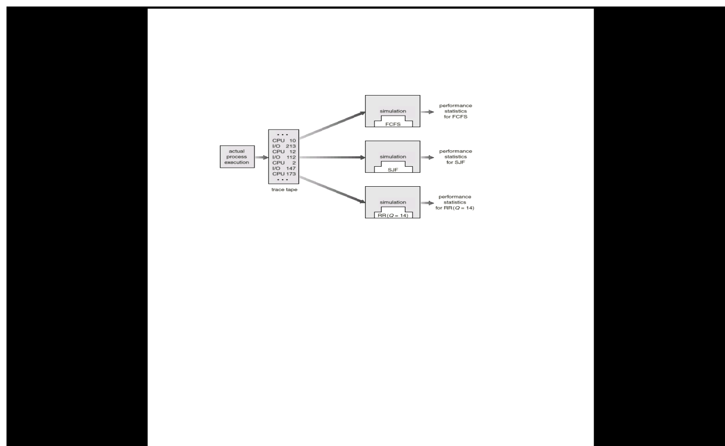
Công thức này được gọi là công thức Little. Công thức Little là đặc biệt có ích vì nó phù hợp cho bất cứ giải thuật định thời và sự phân bổ các quá trình đến.

Chúng ta sử dụng công thức Little để tính một trong ba biến, nếu chúng ta biết hai biến khác. Thí dụ, nếu chúng ta biết có 7 quá trình đến mỗi giây (trung bình) và thường có 14 quá trình trong hàng đợi thì chúng ta có thể tính thời gian chờ đợi trung bình trên mỗi quá trình là 2 giây.

Phân tích hàng đợi có thể có ích trong việc so sánh các giải thuật định thời nhưng nó cũng có một số giới hạn. Hiện nay, các loại giải thuật và sự phân bổ được quản lý là tương đối giới hạn. Tính toán của các giải thuật phức tạp và sự phân bổ là rất khó để thực hiện. Do đó, phân bổ đến và phục vụ thường được định nghĩa không thực tế, nhưng dễ hướng dẫn về mặt tính toán. Thông thường cần thực hiện một số giả định độc lập có thể không chính xác. Do đó, để chúng sẽ có thể tính câu trả lời, các mô hình hàng đợi thường chỉ xấp xỉ hệ thống thật. Vì thế, độ chính xác của các kết quả tính toán có thể là sự nghi vấn.

Mô phỏng

Để đạt được sự đánh giá các giải thuật định thời chính xác hơn, chúng ta có thể dùng mô phỏng (simulations). Mô phỏng liên quan đến lập trình một mô hình hệ thống máy tính. Cấu trúc dữ liệu phần mềm biểu diễn các thành phần quan trọng của hệ thống. Bộ mô phỏng có một biến biểu diễn đồng hồ; khi giá trị của biến này tăng, bộ mô phỏng sửa đổi trạng thái hệ thống để phản ánh các hoạt động của các thiết bị, các quá trình và các bộ định thời. Khi sự mô phỏng thực thi, các thống kê hiển thị năng lực của giải thuật được tập hợp và in ra.



Hình IV-8 Đánh giá các bộ định thời CPU bằng mô phỏng

Dữ liệu để định hướng sự mô phỏng có thể được sinh ra trong nhiều cách. Cách thông dụng nhất dùng bộ sinh số ngẫu nhiên, được lập trình để sinh ra các quá trình, thời gian chu kỳ CPU, đến, đi của quá trình,..dựa trên phân bổ xác suất. Sự phân bổ này có thể được định nghĩa dạng toán học (đồng nhất, hàm mũ, phân bổ Poisson) hay theo kinh nghiệm. Nếu sự phân bổ được định nghĩa theo kinh nghiệm thì các thước đo của hệ thống thật

dưới sự nghiên cứu là lấy được. Các kết quả được dùng để định nghĩa sự phân bổ thật sự các sự kiện trong hệ thống thực và sau đó sự phân bổ này có thể được dùng để định hướng việc mô phỏng.

Tuy nhiên, một mô phỏng hướng phân bổ có thể không chính xác do mối quan hệ giữa các sự kiện tiếp theo trong hệ thống thực. Sự phân bổ thường xuyên hiển thị chỉ bao nhiêu sự kiện xảy ra; nó không hiển thị bất cứ thứ gì về thứ tự xảy ra của chúng. Để sửa chữa vấn đề này, chúng ta dùng băng từ ghi vết (trace tapes). Chúng ta tạo một băng từ ghi vết bằng cách giám sát hệ thống thực, ghi lại chuỗi các sự kiện thật (như hình IV.8). Sau đó, thứ tự này được dùng để định hướng việc mô phỏng. Băng từ ghi vết cung cấp cách tuyệt vời để so sánh chính xác hai giải thuật trên cùng một tập hợp dữ liệu vào thật. Phương pháp này có thể cung cấp các kết quả chính xác cho dữ liệu vào của nó.

Tuy nhiên, mô phỏng có thể rất đắt và thường đòi hỏi hàng giờ máy tính để thực hiện. Một mô phỏng chi tiết hơn cung cấp các kết quả chính xác hơn nhưng cũng yêu cầu nhiều thời gian máy tính hơn. Ngoài ra, các băng từ ghi vết có thể yêu cầu lượng lớn không gian lưu trữ. Cuối cùng, thiết kế, mã, gỡ rối của bộ mô phỏng là một tác vụ quan trọng.

Cài đặt

Ngay cả mô phỏng cũng cho độ chính xác có giới hạn. Chỉ có cách chính xác hoàn toàn để đánh giá giải thuật định thời là mã hóa (code) nó, đặt nó vào trong hệ điều hành và xem nó làm việc như thế nào. Tiếp cận này đặt một giải thuật thật sự vào hệ thống thật để đánh giá dưới điều kiện hoạt động thật sự.

Khó khăn chủ yếu là chi phí của tiếp cận. Chi phí bao gồm không chỉ mã hóa giải thuật và sửa đổi hệ điều hành để hỗ trợ nó cũng như các cấu trúc dữ liệu được yêu cầu mà còn phản ứng của người dùng đối với sự thay đổi liên tục hệ điều hành. Hầu hết người dùng không quan tâm việc xây dựng một hệ điều hành tốt hơn; họ chỉ đơn thuần muốn biết các quá trình của họ thực thi và dùng các kết quả của chúng. Một hệ điều hành thay đổi liên tục không giúp cho người dùng nhận thấy công việc của họ được thực hiện. Một dạng của phương pháp này được dùng phổ biến cho việc cài đặt máy tính mới. Thí dụ, một tiện ích Web mới có thể mô phỏng tải người dùng được phát sinh trước khi nó "sống" (goes live), để xác định bất cứ hiện tượng thắt cổ chai trong tiện ích và để ước lượng bao nhiêu người dùng hệ thống có thể hỗ trợ.

Một khó khăn khác với bất cứ việc đánh giá giải thuật nào là môi trường trong đó giải thuật được dùng sẽ thay đổi. Môi trường sẽ thay đổi không chỉ trong cách thông thường như những chương trình mới được viết và các loại vấn đề thay đổi, mà còn kết quả năng lực của bộ định thời. Nếu các quá trình được cho với độ ưu tiên ngắn thì người dùng có thể tách các quá trình lớn thành tập hợp các quá trình nhỏ hơn. Nếu quá trình giao tiếp được cho độ ưu tiên vượt qua các quá trình không giao tiếp thì người dùng có thể chuyển tới việc dùng giao tiếp.

Thí dụ, trong DEC TOPS-20, hệ thống được phân loại các quá trình giao tiếp và không giao tiếp một cách tự động bằng cách xem lượng nhập/xuất thiết bị đầu cuối. Nếu một quá trình không có nhập hay xuất tới thiết bị đầu cuối trong khoảng thời gian 1 phút thì quá trình được phân loại là không giao tiếp và được di chuyển tới hàng đợi có độ ưu tiên thấp. Chính sách này dẫn đến trường hợp một người lập trình sửa đổi chương trình của mình để viết một ký tự bất kỳ tới thiết bị đầu cuối tại khoảng thời gian đều đặn ít hơn 1 phút. Hệ thống này cho những chương trình này có độ ưu tiên cao mặc dù dữ liệu xuất của thiết bị đầu cuối là hoàn toàn không có ý nghĩa.

Các giải thuật có khả năng mềm dẻo nhất có thể được thay đổi bởi người quản lý hay người dùng. Trong suốt thời gian xây dựng hệ điều hành, thời gian khởi động, thời gian chạy, các biến được dùng bởi các bộ định thời có thể được thay đổi để phản ánh việc sử dụng của hệ thống trong tương lai. Yêu cầu cho việc định thời biểu mềm dẻo là một trường hợp khác mà ở đó sự tách riêng các cơ chế từ chính sách là có ích. Thí dụ, nếu các hóa đơn cần được xử lý và in lập tức nhưng thường được thực hiện như công việc bó có độ ưu tiên thấp, hàng đợi bó được cho tạm thời độ ưu tiên cao hơn. Tuy nhiên, rất ít hệ điều hành chấp nhận loại định thời này.

Tóm tắt

Định thời CPU là một tác vụ chọn một quá trình đang chờ từ hàng đợi sẵn sàng và cấp phát CPU tới nó. CPU được cấp phát tới quá trình được chọn bởi bộ cấp phát.

Định thời đến trước, được phục vụ trước (FCFS) là giải thuật định thời đơn giản nhất, nhưng nó có thể gây các quá trình ngắn chờ các quá trình quá trình quá dài. Định thời ngắn nhất, phục vụ trước (SJF) có thể tối ưu, cung cấp thời gian chờ đợi trung bình ngắn nhất. Cài đặt định thời SJF là khó vì đoán trước chiều dài của chu kỳ CPU kế tiếp là khó. Giải thuật SJF là trường hợp đặc biệt của giải thuật định thời trưng dụng thông thường. Nó đơn giản cấp phát CPU tới quá trình có độ ưu tiên cao nhất. Cả hai định thời độ ưu tiên và SJF có thể gặp phải trở ngại của việc đói tài nguyên.

Định thời quay vòng (RR) là hợp lý hơn cho hệ thống chia sẻ thời gian. Định thời RR cấp phát CPU tới quá trình đầu tiên trong hàng đợi sẵn sàng cho q đơn vị thời gian, ở đây q là định mức thời gian. Sau q đơn vị thời gian, nếu quá trình này không trả lại CPU thì nó bị chiếm và quá trình này được đặt vào đuôi của hàng đợi sẵn sàng. Vấn đề quan trọng là chọn định mức thời gian. Nếu định mức quá lớn, thì định thời RR giảm hơn định thời FCFS ; nếu định mức quá nhỏ thì chi phí định thời trong dạng thời gian chuyển ngữ cảnh trở nên thừa.

Giải thuật FCFS là không ưu tiên; giải thuật RR là ưu tiên. Các giải thuật SJF và ưu tiên có thể ưu tiên hoặc không ưu tiên.

Các giải thuật hàng đợi nhiều cấp cho phép các giải thuật khác nhau được dùng cho các loại khác nhau của quá trình. Chung nhất là hàng đợi giao tiếp ở chế độ hiển thị dùng định thời RR và hàng đợi bó chạy ở chế độ nền dùng định thời FCFS. Hàng đợi phản hồi nhiều cấp cho phép các quá trình di chuyển từ hàng đợi này sang hàng đợi khác.

Vì có nhiều giải thuật định thời sẵn dùng, chúng ta cần các phương pháp để chọn giữa chúng. Các phương pháp phân tích dùng cách thức phân tích toán học để xác định năng lực của giải thuật. Các phương pháp mô phỏng xác định năng lực bằng cách phỏng theo giải thuật định thời trên những mẫu 'đại diện' của quá trình và tính năng lực kết quả.

Luồng
Mục đích Sau khi học xong chương này, người học nắm được những kiến thức sau: - Các khái niệm gán với hệ điều hành đa luồng - Các vấn đề liên quan với lập trình đa luồng - Các ảnh hưởng của luồng tới việc thiết kế hệ điều hành - Cách thức các hệ điều hành hiện đại hỗ trợ luồng

Giới thiệu

Mô hình thực thi trong chương 3 giả sử rằng một quá trình là một chương trình đang thực thi với một luồng điều khiển. Nhiều hệ điều hành hiện đại hiện nay cung cấp các đặc điểm cho một quá trình chứa nhiều luồng (thread) điều khiển. Trong chương này giới thiệu các khái niệm liên quan với các hệ thống máy tính đa luồng, gồm thảo luận Pthread API và luồng Java.

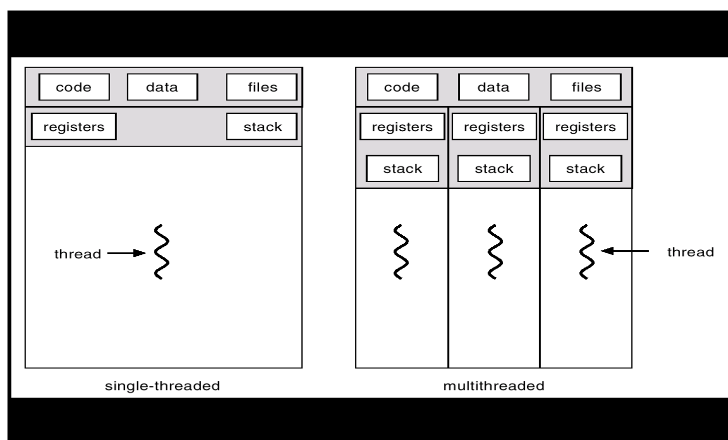
Chúng ta sẽ xem xét nhiều vấn đề có liên quan tới lập trình đa luồng và nó ảnh hưởng như thế nào đến thiết kế của hệ điều hành. Cuối cùng, chúng ta sẽ khám phá nhiều hệ điều hành hiện đại hỗ trợ luồng tại cấp độ nhân như thế nào.

Tổng quan

Một luồng thường được gọi là quá trình nhẹ (lightweight proces-LWP), là một đơn vị cơ bản của việc sử dụng CPU; nó hình thành gồm: một định danh luồng (thread ID), một bộ đếm chương trình, tập thanh ghi và ngăn xếp. Nó chia sẻ với các luồng khác thuộc cùng một quá trình phần mã, phần dữ liệu, và tài nguyên hệ điều hành như các tập tin đang mở và các tín hiệu. Một quá trình truyền thống (hay quá trình nặng) có một luồng điều khiển đơn. Nếu quá trình có nhiều luồng điều khiển, nó có thể thực hiện nhiều hơn một tác vụ tại một thời điểm. Hình VI.1 hiển thị sự khác nhau giữa quá trình đơn luồng và quá trình đa luồng.

Sự cơ động

Nhiều gói phần mềm chạy trên các máy để bàn PC là đa luồng. Điển hình, một ứng dụng được cài đặt như một quá trình riêng rẻ với nhiều luồng điều khiển. Một trình duyệt Web có thể có một luồng hiển thị hình ảnh, văn bản trong khi một luồng khác lấy dữ liệu từ mạng. Một trình soạn thảo văn bản có thể có một luồng hiển thị đồ họa, luồng thứ hai đọc sự bấm phím trên bàn phím từ người dùng, một luồng thứ ba thực hiện việc kiểm tra chính tả và từ vựng chạy trong chế độ nền.



Hình IV-1 Quá trình đơn và đa luồng

Trong những trường hợp cụ thể một ứng dụng đơn có thể được yêu cầu thực hiện nhiều tác vụ đơn. Thí dụ, một trình phục vụ web chấp nhận các yêu cầu khách hàng như trang web, hình ảnh, âm thanh, ..Một trình phục vụ web có thể có nhiều (hàng trăm) khách hàng truy xuất đồng thời nó. Nếu trình phục vụ web chạy như một quá trình đơn luồng truyền thống thì nó sẽ có thể chỉ phục vụ một khách hàng tại cùng thời điểm. Lượng thời gian mà khách hàng phải chờ yêu cầu của nó được phục vụ là rất lớn.

Một giải pháp là có một trình phục vụ chạy như một quá trình đơn chấp nhận các yêu cầu. Khi trình phục vụ nhận một yêu cầu, nó sẽ tạo một quá trình riêng để phục vụ yêu cầu đó. Thật vậy, phương pháp tạo ra quá trình này là cách sử dụng thông thường trước khi luồng trở nên phổ biến. Tạo ra quá trình có ảnh hưởng rất lớn như được trình bày ở chương

trước. Nếu quá trình mới sẽ thực hiện cùng tác vụ như quá trình đã có thì tại sao lại gánh chịu tất cả chi phí đó? Thường sẽ hiệu quả hơn cho một quá trình chứa nhiều luồng phục vụ cùng một mục đích. Tiếp cận này sẽ đa luồng quá trình trình phục vụ web. Trình phục vụ sẽ tạo một luồng riêng lắng nghe các yêu cầu người dùng; khi yêu cầu được thực hiện nó không tạo ra quá trình khác mà sẽ tạo một luồng khác phục vụ yêu cầu.

Luồng cũng đóng một vai trò quan trọng trong hệ thống lời gọi thủ tục xa (remote process call-RPC). Như đã trình bày ở chương trước, RPCs cho phép giao tiếp liên quá trình bằng cách cung cấp cơ chế giao tiếp tương tự như các lời gọi hàm hay thủ tục thông thường. Điển hình, các trình phục vụ RPCs là đa luồng. Khi một trình phục vụ nhận một thông điệp, nó phục vụ thông điệp dùng một luồng riêng. Điều này cho phép phục vụ nhiều yêu cầu đồng hành.

Thuận lợi

Những thuận lợi của lập trình đa luồng có thể được chia làm bốn loại:

- Sự đáp ứng: đa luồng một ứng dụng giao tiếp cho phép một chương trình tiếp tục chạy thậm chí nếu một phần của nó bị khóa hay đang thực hiện một thao tác dài, do đó gia tăng sự đáp ứng đối với người dùng. Thí dụ, một trình duyệt web vẫn có thể đáp ứng người dùng bằng một luồng trong khi một ảnh đang được nạp bằng một luồng khác.
- Chia sẻ tài nguyên: Mặc định, các luồng chia sẻ bộ nhớ và các tài nguyên của các quá trình mà chúng thuộc về. Thuận lợi của việc chia sẽ mã là nó cho phép một ứng dụng có nhiều hoạt động của các luồng khác nhau nằm trong cùng không gian địa chỉ.
- Kinh tế: cấp phát bộ nhớ và các tài nguyên cho việc tạo các quá trình là rất đắt. Vì các luồng chia sẻ tài nguyên của quá trình mà chúng thuộc về nên nó kinh tế hơn để tạo và chuyển ngữ cảnh giữa các luồng. Khó để đánh giá theo kinh nghiệm sự khác biệt chi phí cho việc tạo và duy trì một quá trình hơn một luồng, nhưng thường nó sẽ mất nhiều thời gian để tạo và quản lý một quá trình hơn một

luồng. Trong Solaris 2, tạo một quá trình chậm hơn khoảng 30 lần tạo một luồng và chuyển đổi ngữ cảnh chậm hơn 5 lần.

- Sử dụng kiến trúc đa xử lý: các lợi điểm của đa luồng có thể phát huy trong kiến trúc đa xử lý, ở đó mỗi luồng thực thi song song trên một bộ xử lý khác nhau. Một quá trình đơn luồng chỉ có thể chạy trên một CPU. Đa luồng trên một máy nhiều CPU gia tăng tính đồng hành. Trong kiến trúc đơn xử lý, CPU thường chuyển đổi qua lại giữa mỗi luồng quá nhanh để tạo ra hình ảnh của sự song song nhưng trong thực tế chỉ một luồng đang chạy tại một thời điểm.

Luồng người dùng và luồng nhân

Chúng ta vừa mới thảo luận là xem xét luồng như một chiều hướng chung. Tuy nhiên, hỗ trợ luồng được cung cấp hoặc ở cấp người dùng, cho các luồng người dùng hoặc ở cấp nhân, cho các luồng nhân.

- Luồng người dùng: được hỗ trợ dưới nhân và được cài đặt bởi thư viện luồng tại cấp người dùng. Thư viện cung cấp hỗ trợ cho việc tạo luồng, lập thời biểu, và quản lý mà không có sự hỗ trợ từ nhân. Vì nhân không biết các luồng cấp người dùng, tất cả việc tạo luồng và lập thời biểu được thực hiện trong không gian người dùng mà không cần sự can thiệp của nhân. Do đó, các luồng cấp người dùng thường tạo và quản lý nhanh, tuy nhiên chúng cũng có những trở ngại. Thí dụ, nếu nhân là đơn luồng thì bất cứ luồng cấp người dùng thực hiện một lời gọi hệ thống nghẽn sẽ làm cho toàn bộ quá trình bị nghẽn, thậm chí nếu các luồng khác sẵn dùng để chạy trong ứng dụng. Các thư viện luồng người dùng gồm các luồng POSIX Pthreads, Mach C-threads và Solaris 2 UI-threads.
- Luồng nhân: được hỗ trợ trực tiếp bởi hệ điều hành. Nhân thực hiện việc tạo luồng, lập thời biểu, và quản lý không gian nhân. Vì quản lý luồng được thực hiện bởi hệ điều hành, luồng nhân thường tạo và quản lý chậm hơn luồng người dùng. Tuy nhiên, vì nhân được quản lý các luồng nếu một luồng thực hiện lời gọi hệ thống nghẽn, nhân có thể lập thời biểu một luồng khác trong ứng dụng

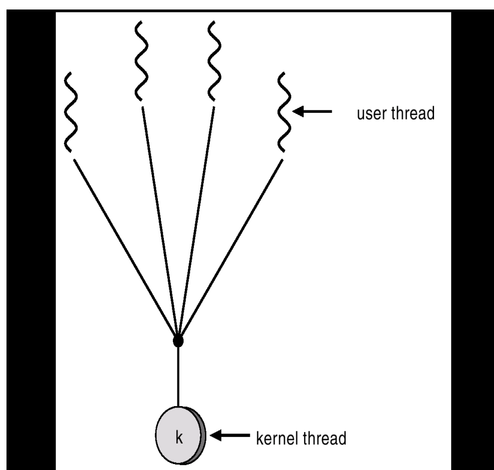
thực thi. Trong môi trường đa xử lý, nhân có thể lập thời biểu luồng trên một bộ xử lý khác. Hầu hết các hệ điều hành hiện nay như Windows NT, Windows 2000, Solaris 2, BeOS và Tru64 UNIX (trước Digital UNIX)-hỗ trợ các luồng nhân.

Mô hình đa luồng

Nhiều hệ thống cung cấp sự hỗ trợ cả hai luồng nhân và luồng người dùng nên tạo ra nhiều mô hình đa luồng khác nhau. Chúng ta sẽ xem xét ba loại cài đặt luồng thông thường

Mô hình nhiều-một

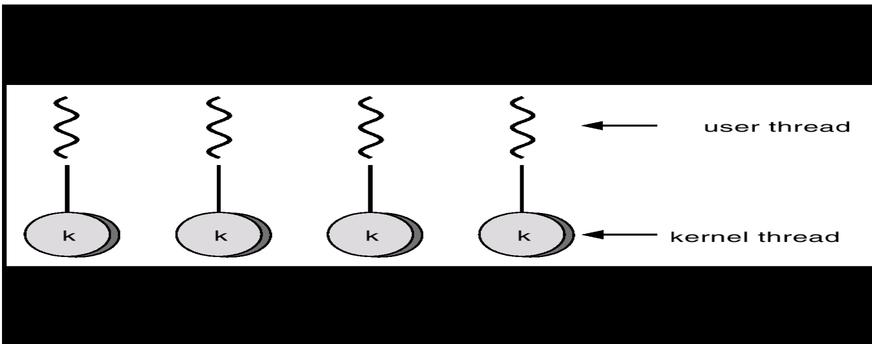
Mô hình nhiều-một (như hình IV.2) ánh xạ nhiều luồng cấp người dùng tới một luồng cấp nhân. Quản lý luồng được thực hiện trong không gian người dùng vì thế nó hiệu quả nhưng toàn bộ quá trình sẽ bị khóa nếu một luồng thực hiện lời gọi hệ thống khóa. Vì chỉ một luồng có thể truy xuất nhân tại một thời điểm nên nhiều luồng không thể chạy song song trên nhiều bộ xử lý. Green threads-một thư viện luồng được cài đặt trên các hệ điều hành không hỗ trợ luồng nhân dùng mô hình nhiều-một.



Hình IV-2-Mô hình nhiều-một

Mô hình một-một

Mô hình một-một (hình IV.3) ánh xạ mỗi luồng người dùng tới một luồng nhân. Nó cung cấp khả năng đồng hành tốt hơn mô hình nhiều-một bằng cách cho một luồng khác chạy khi một luồng thực hiện lời gọi hệ thống nghẽn; nó cũng cho phép nhiều luồng chạy song song trên các bộ xử lý khác nhau. Chỉ có một trở ngại trong mô hình này là tạo luồng người dùng yêu cầu tạo một luồng nhân tương ứng. Vì chi phí cho việc tạo luồng nhân có thể đè nặng lên năng lực thực hiện của ứng dụng, các cài đặt cho mô hình này giới hạn số luồng được hỗ trợ bởi hệ thống. Windows NT, Windows 2000 và OS/2 cài đặt mô hình một-một này.

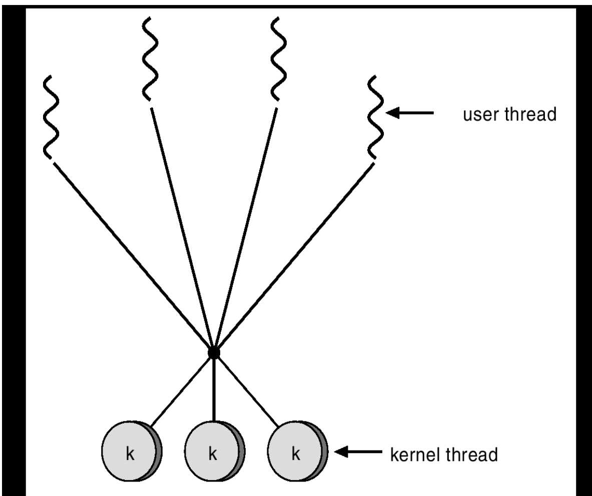


Hình IV-3-Mô hình một-một

Mô hình nhiều-nhiều

Mô hình nhiều-nhiều (như hình VI.4) đa hợp nhiều luồng cấp người dùng tới số lượng nhỏ hơn hay bằng các luồng nhân. Số lượng các luồng nhân có thể được xác định hoặc một ứng dụng cụ thể hay một máy cụ thể (một ứng dụng có thể được cấp nhiều luồng nhân trên một bộ đa xử lý hơn trên một bộ đơn xử lý). Trong khi mô hình nhiều-một cho phép người phát triển tạo nhiều luồng người dùng như họ muốn, thì đồng hành thật sự là không đạt được vì nhân có thể lập thời biểu chỉ một luồng tại một thời điểm. Mô hình một-một cho phép đồng hành tốt hơn nhưng người phát triển phải cẩn thận không tạo ra quá nhiều luồng trong

một ứng dụng. Mô hình nhiều-nhiều gặp phải một trong hai vấn đề khiếm khuyết: người phát triển có thể tạo nhiều luồng người dùng khi cần thiết và các luồng nhân tương ứng có thể chạy song song trên một bộ đa xử lý. Khi một luồng thực hiện một lời gọi hệ thống khóa, nhân có thể lập thời biểu một luồng khác thực thi. Solaris 2, IRIX, HP-UX, và Tru64 UNIX hỗ trợ mô hình này.



Hình IV-4-Mô hình nhiều-nhiều

Cấp phát luồng

Trong phần này chúng ta thảo luận các cấp phát xem xét với các chương trình đa luồng.

Lời gọi hệ thống fork và exec

Trong chương trước chúng ta mô tả lời gọi hệ thống fork được dùng để tạo một quá trình bản sao riêng như thế nào. Trong một chương trình đa luồng, ngữ nghĩa của các lời gọi hệ thống fork và exec thay đổi. Nếu một luồng trong lời gọi chương trình fork thì quá trình mới sao chép lại quá trình tất cả luồng hay là một quá trình đơn luồng mới? Một số hệ thống

UNIX chọn hai ấn bản fork, một sao chép lại tất cả luồng và một sao chép lại chỉ luồng được nạp lên lời gọi hệ thống fork. Lời gọi hệ thống exec điển hình thực hiện công việc trong cùng một cách như được mô tả trong chương trước. Nghĩa là, nếu một luồng nạp lời gọi hệ thống exec, chương trình được xác định trong tham số exec sẽ thay thế toàn bộ quá trình-chứa tất cả luồng và các quá trình tải nhẹ.

Việc sử dụng hai ấn bản fork phụ thuộc vào ứng dụng. Nếu exec bị hủy tức thì sau khi phân nhánh (forking) thì sự sao chép lại tất cả luồng là không cần thiết khi chương trình được xác định trong các tham số exec sẽ thay thế quá trình. Trong trường hợp này, việc sao chép lại chỉ gọi luồng hợp lý. Tuy nhiên, nếu quá trình riêng biệt này không gọi exec sau khi phân nhánh thì quá trình riêng biệt này nên sao chép lại tất cả luồng.

Sự hủy bỏ luồng

Hủy một luồng là một tác vụ kết thúc một luồng trước khi nó hoàn thành.Thí dụ, nếu nhiều luồng đang tìm kiếm đồng thời thông qua một cơ sở dữ liệu và một luồng trả về kết quả, các luồng còn lại có thể bị hủy. Một trường hợp khác có thể xảy ra khi người dùng nhấn một nút trên trình duyệt web để dừng trang web đang được tải. Thường một trang web được tải trong một luồng riêng. Khi người dùng nhấn nút stop, luồng đang nạp trang bị hủy bỏ.

Một luồng bị hủy thường được xem như luồng đích. Sự hủy bỏ một luồng đích có thể xảy ra hai viễn cảnh khác nhau:

- Hủy bất đồng bộ: một luồng lập tức kết thúc luồng đích
- Hủy trì hoãn: luồng đích có thể kiểm tra định kỳ nếu nó sắp kết thúc, cho phép luồng đích một cơ hội tự kết thúc trong một cách có thứ tự.

Sự khó khăn của việc hủy này xảy ra trong những trường hợp khi tài nguyên được cấp phát tới một luồng bị hủy hay một luồng bị hủy trong khi việc cập nhật dữ liệu xảy ra giữa chừng, nó đang chia sẻ với các

luồng khác. Điều này trở nên đặc biệt khó khăn với sự hủy bất đồng bộ. Hệ điều hành thường đòi lại tài nguyên hệ thống từ luồng bị hủy nhưng thường nó sẽ không đòi lại tất cả tài nguyên. Do đó, việc hủy một luồng bất đồng bộ có thể không giải phóng hết tài nguyên hệ thống cần thiết.

Một chọn lựa khác, sự hủy trì hoãn thực hiện bằng một luồng báo hiệu rằng một luồng đích bị hủy. Tuy nhiên, sự hủy sẽ xảy ra chỉ khi luồng đích kiểm tra để xác định nếu nó được hủy hay không. Điều này cho phép một luồng kiểm tra nếu nó sẽ bị hủy tại điểm nó có thể an toàn bị hủy. Pthreads gọi những điểm như thế là các điểm hủy (cancellation points).

Hầu hết hệ điều hành cho phép một quá trình hay một luồng bị hủy bất đồng bộ. Tuy nhiên, Pthread API cung cấp sự hủy trì hoãn. Điều này có nghĩa rằng một hệ điều hành cài đặt Pthread API sẽ cho phép sự hủy có trì hoãn.

Tín hiệu quản lý

Một tín hiệu (signal) được dùng trong hệ điều hành UNIX thông báo một sự kiện xác định xảy ra. Một tín hiệu có thể được nhận hoặc đồng bộ hoặc bất đồng bộ phụ thuộc mã và lý do cho sự kiện đang được báo hiệu. Một tín hiệu hoặc đồng bộ hoặc bất đồng bộ đều theo sau cùng mẫu:

- Tín hiệu được phát sinh bởi sự xảy ra của một sự kiện xác định.
- Tín hiệu được phát sinh được phân phát tới một quá trình.
- Khi được phân phát xong, tín hiệu phải được quản lý.

Một thí dụ của tín hiệu đồng bộ gồm một truy xuất bộ nhớ không hợp lệ hay chia cho 0. Trong trường hợp này, nếu một chương trình đang chạy thực hiện một trong các hoạt động này, một tín hiệu được phát sinh. Các tín hiệu đồng bộ được phân phát tới cùng một quá trình thực hiện thao tác gây ra tín hiệu (do đó lý do chúng được xem đồng bộ).

Khi một tín hiệu được phát sinh bởi một sự kiện bên ngoài tới một quá trình đang chạy, quá trình đó nhận tín hiệu bất đồng bộ. Thí dụ, tín hiệu kết thúc quá trình với phím xác định (như <control><C>) hay thời gian hết hạn. Điển hình, một tín hiệu bất đồng bộ được gởi tới quá trình khác.

Mỗi tín hiệu có thể được quản lý bởi một trong hai bộ quản lý:

- Bộ quản lý tín hiệu mặc định
- Bộ quản lý tín hiệu được định nghĩa bởi người dùng

Mỗi tín hiệu có một bộ quản lý tín hiệu mặc định được thực thi bởi nhân khi quản lý tín hiệu. Hoạt động mặc định có thể được ghi đè bởi một hàm quản lý tín hiệu được định nghĩa bởi người dùng. Trong trường hợp này, hàm được định nghĩa bởi người dùng được gọi để quản lý tín hiệu hơn là hoạt động mặc định. Cả hai tín hiệu đồng bộ và bất đồng bộ có thể được quản lý trong các cách khác nhau. Một số tín hiệu có thể được bỏ qua (như thay đổi kích thước của cửa sổ); các tín hiệu khác có thể được quản lý bằng cách kết thúc chương trình (như truy xuất bộ nhớ không hợp lệ).

Quản lý tín hiệu trong những chương trình đơn luồng không phức tạp; các tín hiệu luôn được phân phát tới một quá trình. Tuy nhiên, phân phát tín hiệu là phức tạp hơn trong những chương trình đa luồng, như một quá trình có nhiều luồng. Một tín hiệu nên được phân phát ở đâu?

Thông thường, các tuỳ chọn sau tồn tại:

- Phân phát tín hiệu tới luồng mà tín hiệu áp dụng
- Phân phát tín hiệu tới mỗi luồng trong quá trình.
- Phân phát tín hiệu tới các luồng cụ thể trong quá trình.
- Gán một luồng xác định để nhận tất cả tín hiệu cho quá trình.

Phương pháp cho việc phân phát tín hiệu phụ thuộc vào loại tín hiệu được phát sinh. Thí dụ, các tín hiệu đồng bộ cần được phân phát tới luồng đã phát sinh ra tín hiệu và không phân phát tới luồng nào khác trong quá trình. Tuy nhiên, trường hợp với tín hiệu bất đồng bộ là không rõ

ràng. Một số tín hiệu bất đồng bộ - như tín hiệu kết thúc một quá trình (thí dụ:<control><c>)- nên được gởi tới tất cả luồng. Một số ấn bản đa luồng của UNIX cho phép một luồng xác định tín hiệu nào sẽ được chấp nhận và tín hiệu nào sẽ bị khoá. Do đó, một vài tín hiệu bất đồng bộ có thể được phân phát tới chỉ các luồng không khoá tín hiệu. Tuy nhiên, vì tín hiệu cần được quản lý chỉ một lần, điển hình một tín hiệu được phân phát chỉ luồng đầu tiên được tìm thấy trong một luồng mà không nghẽn tín hiệu. Solaris 2 cài đặt bốn tuỳ chọn; nó tạo một luồng xác định trong mỗi quá trình cho quản lý tín hiệu. Khi một tín hiệu bất đồng bộ được gởi tới một quá trình, nó được gởi tới luồng xác định, sau đó nó phân phát tín hiệu tới luồng đầu tiên không khoá tín hiệu.

Mặc dù Windows 2000 không cung cấp rõ sự hỗ trợ tín hiệu, nhưng chúng có thể được mô phỏng sử dụng lời gọi thủ tục bất đồng bộ (asynchronous produce calls-APC). Tiện ích APC cho phép luồng người dùng xác định hàm được gọi khi luồng người dùng nhận thông báo về một sự kiện xác định. Như được hiển thị bởi tên của nó, một APC rất giống tín hiệu bất đồng bộ trong UNIX. Tuy nhiên, UNIX phải đấu tranh với cách giải quyết tín hiệu trong môi trường đa luồng, phương tiện APC phức tạp hơn như một APC được phân phát tới luồng xác định hơn quá trình.

Nhóm luồng

Trong phần VI.3, chúng ta mô tả kịch bản đa luồng của một trình phục vụ web. Trong trường hợp này, bất cứ khi nào trình phục vụ nhận một yêu cầu, nó tạo một luồng riêng để phục vụ yêu cầu đó. Ngược lại, tạo một luồng riêng thật sự cao hơn tạo một quá trình riêng, dù sao một trình phục vụ đa luồng có thể phát sinh vấn đề. Quan tâm đầu tiên là lượng thời gian được yêu cầu để tạo luồng trước khi phục vụ yêu cầu, và lượng thời gian xoá luồng khi nó hoàn thành. Vấn đề thứ hai là vấn đề khó giải quyết hơn: nếu chúng ta cho phép tất cả yêu cầu đồng hành được phục vụ trong một luồng mới, chúng ta không thay thế giới hạn trên số lượng luồng hoạt động đồng hành trong hệ thống. Những luồng

không giới hạn có thể làm cạn kiệt tài nguyên hệ thống, như thời gian CPU và bộ nhớ. Một giải pháp cho vấn đề này là sử dụng nhóm luồng.

Ý tưởng chung nằm sau nhóm luồng là tạo số lượng luồng tại thời điểm khởi động và đặt chúng vào nhóm, nơi chúng ngồi và chờ công việc. Khi một trình phục vụ nhận một yêu cầu, chúng đánh thức một luồng từ nhóm- nếu một luồng sẵn dùng – truyền nó yêu cầu dịch vụ. Một khi luồng hoàn thành dịch vụ của nó, nó trả về nhóm đang chờ công việc kế. Nếu nhóm không chứa luồng sẵn dùng, trình phục vụ chờ cho tới khi nó rảnh.

Nói cụ thể, các lợi ích của nhóm luồng là:

1. Thường phục vụ yêu cầu nhanh hơn với luồng đã có hơn là chờ để tạo luồng.
2. Một nhóm luồng bị giới hạn số lượng luồng tồn tại bất kỳ thời điểm nào. Điều này đặc biệt quan trọng trên những hệ thống không hỗ trợ số lượng lớn các luồng đồng hành.

Số lượng luồng trong nhóm có thể được đặt theo kinh nghiệm (heuristics) dựa trên các yếu tố như số CPU trong hệ thống, lượng bộ nhớ vật lý và số yêu cầu khách hàng đồng hành. Kiến trúc nhóm luồng tinh vi hơn có thể tự điều chỉnh số lượng luồng trong nhóm dựa theo các mẫu sử dụng. Những kiến trúc như thế cung cấp lợi điểm xa hơn của các nhóm luồng nhỏ hơn-do đó tiêu tốn ít bộ nhớ hơn-khi việc nạp trên hệ thống là chậm.

Dữ liệu đặc tả luồng

Các luồng thuộc một quá trình chia sẻ dữ liệu của quá trình. Thật vậy, chia sẻ dữ liệu này cung cấp một trong những lợi điểm của lập trình đa luồng. Tuy nhiên, mỗi luồng có thể cần bản sao dữ liệu xác định của chính nó trong một vài trường hợp. Chúng ta sẽ gọi dữ liệu như thế là dữ liệu đặc tả luồng. Thí dụ, trong một hệ thống xử lý giao dịch, chúng ta có thể phục vụ mỗi giao dịch trong một luồng. Ngoài ra, mỗi giao dịch có

thể được gán một danh biểu duy nhất. Để gán mỗi luồng với định danh duy nhất của nó chúng ta có thể dùng dữ liệu đặc tả dữ liệu. Hầu hết thư viện luồng gồm Win32 và Pthread – cung cấp một số biểu mẫu hỗ trợ cho dữ liệu đặc tả luồng. Java cũng cung cấp sự hỗ trợ như thế.

Pthreads

Pthreads tham chiếu tới chuẩn POSIX (IEEE 1003.1c) định nghĩa API cho việc tạo và đồng bộ luồng. Đây là một đặc tả cho hành vi luồng không là một cài đặt. Người thiết kế hệ điều hành có thể cài đặt đặc tả trong cách mà họ muốn. Thông thường, các thư viện cài đặt đặc tả Pthread bị giới hạn đối với các hệ thống dựa trên cơ sở của UNIX như Solaris 2. Hệ điều hành Windows thường không hỗ trợ Pthreads mặc dù các ấn bản shareware là sẵn dùng trong phạm vi công cộng.

Trong phần này chúng ta giới thiệu một số Pthread API như một thí dụ cho thư viện luồng cấp người dùng. Chúng ta sẽ xem nó như thư viện cấp người dùng vì không có mối quan hệ khác biệt giữa một luồng được tạo dùng Pthread và luồng được gắn với nhân. Chương trình C hiển thị trong hình dưới đây, mô tả một Pthread API cơ bản để xây dựng một chương trình đa luồng.

Chương trình hiển thị trong hình tạo một luồng riêng xác định tính tổng của một số nguyên không âm. Trong chương trình Pthread, các luồng riêng bắt đầu thực thi trong một hàm xác định. Trong hình, đây là một hàm runner. Khi chương trình này bắt đầu, một luồng riêng điều khiển bắt đầu trong main. Sau khi khởi tạo, main tạo ra luồng thứ hai bắt đầu điều khiển trong hàm runner.

Bây giờ chúng ta sẽ cung cấp tổng quan của chương trình này chi tiết hơn. Tất cả chương trình Pthread phải chứa tập tin tiêu đề pthread.h. pthread_t tid khai báo danh biểu cho luồng sẽ được tạo. Mỗi luồng có một tập các thuộc tính gồm kích thước ngăn xếp và thông tin định thời. Khai báo pthread_attr_t attr hiện diện các thuộc tính cho luồng. Chúng ta sẽ thiết lập các thuộc tính trong gọi hàm pthread_attr_init(&attr). Vì chúng ta không thiết lập rõ thuộc tính, chúng ta sẽ dùng thuộc tính mặc

định được cung cấp. Một luồng riêng được tạo với lời gọi hàm pthread_create. Ngoài ra, để truyền định danh của luồng và các thuộc tính cho luồng, chúng ta cũng truyền tên của hàm, nơi một luồng mới sẽ bắt đầu thực thi, trong trường hợp này là hàm runner. Cuối cùng chúng ta sẽ truyền số nguyên được cung cấp tại dòng lệnh, argv[1].

Tại điểm này, chương trình có hai luồng: luồng khởi tạo trong main và luồng thực hiện việc tính tổng trong hàm runner. Sau khi tạo luồng thứ hai, luồng main sẽ chờ cho luồng runner hoàn thành bằng cách gọi hàm pthread_join. Luồng runner sẽ hoàn thành khi nó gọi hàm pthread_exit.

```
#include<pthread>

#include<stdio.h>

int sum: /*Dữ liệu này được chia sẻ bởi thread(s)*/

void *runner(void *param); /*luồng*/

main(int argc, char *argv[])

{

pthread_t tid; /*định danh của luồng*/

pthread_attr_t attr; /*tập hợp các thuộc tính*/

if(argc !=2){

fprintf(stderr, “usage: a.out <integer value>”);

exit();

}

if (atoi(argv[1] < 0)){

fprintf(stderr,”%d must be >= 0 \n”, atoi(argv[1]));
```

```
exit();
}
/*lấy các thuộc tính mặc định*/
pthread_attr_init(&attr);
/*tạo một luồng*/
pthread_create(&tid,&attr,runner, argv[1]);
/*bây giờ chờ luồng kết thúc*/
pthread_join(tid,NULL);
printf("sum = %d\n",sum);
/*Luồng sẽ bắt đầu điều khiển trong hàm này*/
void *runner(void *param)
{
int upper = atoi(param);
int i;
sum = 0;
if (upper > 0){
sum+= i;
}
pthread_exit(0);
}
```

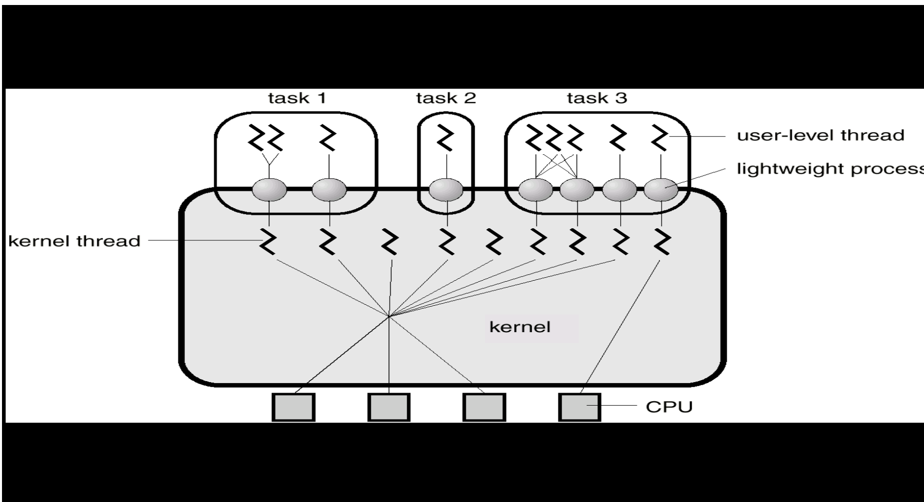
}

Hình IV-5-Chương trình C đa luồng dùng Pthread API

Luồng Solaris 2

Solaris 2 là một ấn bản của UNIX với hỗ trợ luồng tại cấp độ nhân và cấp độ người dùng, đa xử lý đối xứng (SMP) và định thời thời thực. Solaris 2 cài đặt Pthread API hỗ trợ luồng cấp người dùng với thư viện chứa APIs cho việc tạo và quản lý luồng (được gọi luồng UI). Sự khác nhau giữa hai thư viện này rất lớn, mặc dù hầu hết người phát triển hiện nay chọn thư viện Pthread. Solaris 2 cũng định nghĩa một cấp độ luồng trung gian. Giữa luồng cấp nhân và cấp người dùng là các quá trình nhẹ (lightweight process- LWPs). Mỗi quá trình chứa ít nhất một LWP. Thư viện luồng đa hợp luồng người dùng trên nhóm LWP cho quá trình và chỉ luồng cấp người dùng hiện được nối kết tới một LWP hoàn thành công việc. Các luồng còn lại bị khoá hoặc chờ cho một LWP mà chúng có thể thực thi trên nó.

Luồng cấp nhân chuẩn thực thi tất cả thao tác trong nhân. Mỗi LWP có một luồng cấp nhân, và một số luồng cấp nhân (kernel) chạy trên một phần của nhân và không có LWP kèm theo (thí dụ, một luồng phục vụ yêu cầu đĩa). Các luồng cấp nhân chỉ là những đối tượng được định thời trong hệ thống. Solaris 2 cài mô hình nhiều-nhiều; toàn bộ hệ thống luồng của nó được mô tả trong hình dưới đây:



Hình IV-6-Luồng Solaris 2

Các luồng cấp người dùng có thể giới hạn hay không giới hạn. Một luồng cấp người dùng giới hạn được gán vĩnh viễn tới một LWP. Chỉ luồng đó chạy trên LWP và yêu cầu LWP có thể được tận hiến tới một bộ xử lý đơn (xem luồng trái nhất trong hình trên). Liên kết một luồng có ích trong trường hợp yêu cầu thời gian đáp ứng nhanh, như ứng dụng thời thực. Một luồng không giới hạn gán vĩnh viễn tới bất kỳ LWP nào. Tất cả các luồng không giới hạn được đa hợp trong một nhóm cac LWP sẵn dùng cho ứng dụng. Các luồng không giới hạn là mặc định. Solaris 8 cũng cung cấp một thư viện luồng thay đổi mà mặc định chúng liên kết tới tất cả các luồng với một LWP.

Xem xét hệ thống trong hoạt động: bất cứ một quá trình nào có thể có nhiều luồng người dùng. Các luồng cấp người dùng này có thể được định thời và chuyển đổi giữa LWPs bởi thư viện luồng không có sự can thiệp của nhân. Các luồng cấp người dùng cực kỳ hiệu quả vì không có sự hỗ trợ nhân được yêu cầu cho việc tạo hay huỷ, hay thư viện luồng chuyển ngữ cảnh từ luồng người dùng này sang luồng khác.

Mỗi LWP được nối kết tới chính xác một luồng cấp nhân, ngược lại mỗi luồng cấp người dùng là độc lập với nhân. Nhiều LWPs có thể ở trong một quá trình, nhưng chúng được yêu cầu chỉ khi luồng cần giao tiếp với một nhân. Thí dụ, một LWP được yêu cầu mỗi luồng có thể khoá đồng hành trong lời gọi hệ thống. Xem xét năm tập tin khác nhau-đọc các yêu

cầu xảy ra cùng một lúc. Sau đó, năm LWPs được yêu cầu vì chúng đang chờ hoàn thành nhập/xuất trong nhân. Nếu một tác vụ chỉ có bốn LWPs thì yêu cầu thứ năm sẽ không phải chờ một trong những LWPs để trả về từ nhân. Bổ sung một LWP thứ sáu sẽ không đạt được gì nếu chỉ có đủ công việc cho năm.

Các luồng nhân được định thời bởi bộ lập thời biểu của nhân và thực thi trên một hay nhiều CPU trong hệ thống. Nếu một luồng nhân khoá (trong khi chờ một thao tác nhập/xuất hoàn thành), thì bộ xử lý rảnh để thực thi luồng nhân khác. Nếu một luồng bị khoá đang chạy trên một phần của LWP thì LWP cũng khoá. Ở trên vòng, luồng cấp người dùng hiện được gán tới LWP cũng bị khoá. Nếu một quá trình có nhiều hơn một LWP thì nhân có thể định thời một LWP khác.

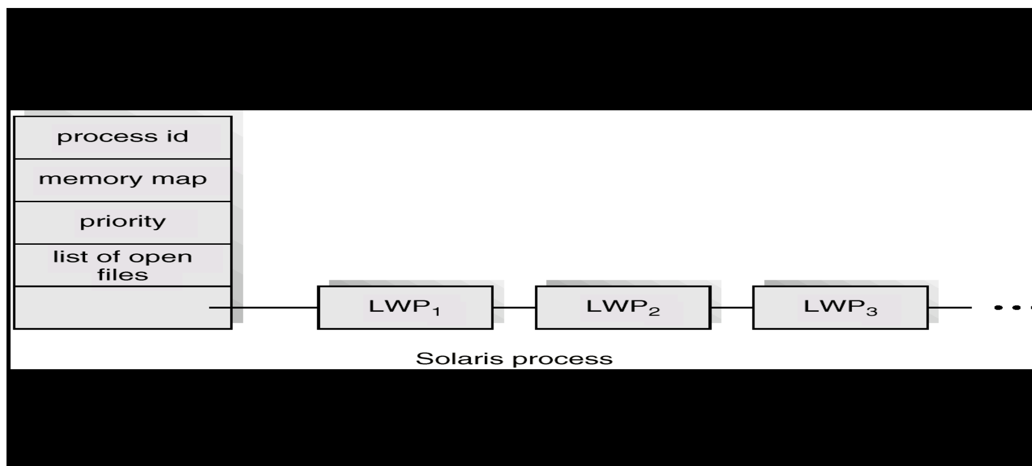
Thư viện luồng tự động thay đổi số lượng LWPs trong nhóm để đảm bảo năng lực thực hiện tốt nhất cho ứng dụng. Thí dụ, nếu tất cả LWPs trong một quá trình bị khoá bởi những luồng có thể chạy thì thư viện tự tạo một LWP khác được gán tới một luồng đang chờ. Do đó, một chương trình được ngăn chặn từ một chương trình khác bởi sự nghèo nàn của những LWPs không bị khoá. LWPs là những tài nguyên nhân đắt để duy trì nếu chúng không được dùng. Thư viện luồng "ages" LWPs và xoá chúng khi chúng không được dùng cho khoảng thời gian dài, điển hình khoảng 5 phút.

Các nhà phát triển dùng những cấu trúc dữ liệu cài đặt luồng trên Solaris 2:

- Luồng cấp người dùng chứa một luồng ID; tập thanh ghi (gồm một bộ đếm chương trình và con trỏ ngăn xếp); ngăn xếp; và độ ưu tiên (được dùng bởi thư viện cho mục đích định thời). Không có cấu trúc dữ liệu nào là tài nguyên nhân; tất cả chúng tồn tại trong không gian người dùng.
- Một LWP có một tập thanh ghi cho luồng cấp nhân nó đang chạy cũng như bộ nhớ và thông tin tính toán. Một LWP là một cấu trúc dữ liệu nhân và nó nằm trong không gian nhân

- Một luồng nhân chỉ có một cấu trúc dữ liệu nhân và ngăn xếp. Cấu trúc dữ liệu gồm bản sao các thanh ghi nhân, con trỏ tới LWP mà nó được gán, độ ưu tiên và thông tin định thời.

Mỗi quá trình trong Solaris 2 gồm nhiều thông tin được mô tả trong khối điều khiển quá trình (Process Control Block-PCB). Trong thực tế, một quá trình Solaris 2 chứa một định danh quá trình (Process ID-PID); bản đồ bộ nhớ; danh sách các tập tin đang mở, độ ưu tiên; và con trỏ của các luồng nhân vơi quá trình (Hình).



Hình IV-7-Quá trình Solaris 2

Luồng Windows 2000

Windows 2000 cài đặt Win32 API. Win32 API là một API chủ yếu cho họ hệ điều hành Windows (Windows 95/98/NT và Windows 2000). Thực vậy, những gì được đề cập trong phần phần này áp dụng tới họ hệ điều hành này

Ứng dụng Windows chạy như một quá trình riêng rẻ nơi mỗi quá trình có thể chứa một hay nhiều luồng. Windows 2000 dùng ánh xạ một-một nơi mà mỗi luồng cấp người dùng ánh xạ tới luồng nhân được liên kết tới.Tuy nhiên, Windows cũng cung cấp sự hỗ trợ cho một thư viện có cấu trúc (fiber library) cung cấp chức năng của mô hình nhiều-nhiều. Mỗi

luồng thuộc về một quá trình có thể truy xuất một không gian địa chỉ ảo của quá trình.

Những thành phần thông thường của một luồng gồm:

- ID của luồng định danh duy nhất luồng
- Tập thanh ghi biểu diễn trạng thái của bộ xử lý
- Ngăn xếp người dùng khi luồng đang chạy ở chế độ người dùng. Tương tự, mỗi luồng cũng có một ngăn xếp nhân được dùng khi luồng đang chạy trong chế độ nhân
- Một vùng lưu trữ riêng được dùng bởi nhiều thư viện thời gian thực và thự viện liên kết động (DLLs).

Tập thanh ghi, ngăn xếp và vùng lưu trữ riêng được xem như ngữ cảnh của luồng và được đặc tả kiến trúc tới phần cứng mà hệ điều hành chạy trên đó. Cấu trúc dữ liệu chủ yếu của luồng gồm:

- RTHREAD (executive thread block-khối luồng thực thi).
- KTHREAD (kernel thread-khối luồng nhân)
- TEB (thread environment block-khối môi trường luồng)

Các thành phần chủ yếu của RTHREAD gồm một con trỏ chỉ tới quá trình nào luồng thuộc về và địa chỉ của thủ tục mà luồng bắt đầu điều khiển trong đó. ETHREAD cũng chứa một con trỏ chỉ tới KTHREAD tương ứng.

KTHREAD gồm thông tin định thời và đồng bộ hóa cho luồng. Ngoài ra, KTHREAD chứa ngăn xếp nhân (được dùng khi luồng đang chạy trong chế độ nhân) và con trỏ chỉ tới TEB.

ETHREAD và KTHREAD tồn tại hoàn toàn ở không gian nhân; điều này có nghĩa chỉ nhân có thể truy xuất chúng. TEB là cấu trúc dữ liệu trong không gian người dùng được truy xuất khi luồng đang chạy ở chế độ người dùng. Giữa những trường khác nhau, TEB chứa ngăn xếp người dùng và một mảng cho dữ liệu đặc tả luồng (mà Windows gọi là lưu trữ cục bộ luồng)

Luồng Linux

Nhân Linux được giới thiệu trong ấn bản 2.2. Linux cung cấp một lời gọi hệ thống fork với chức năng truyền thống là tạo bản sao một quá trình. Linux cũng cung cấp lời gọi hệ thống clone mà nó tương tự như tạo một luồng. clone có hành vi rất giống như fork, ngoại trừ thay vì tạo một bản sao của quá trình gọi, nó tạo một quá trình riêng chia sẻ không gian địa chỉ của quá trình gọi. Nó chấm dứt việc chia sẻ không gian địa chỉ của quá trình cha mà một tác vụ được nhân bản đối xử giống rất nhiều một luồng riêng rẻ.

Chia sẻ không gian địa chỉ được cho phép vì việc biểu diễn của một quá trình trong nhân Linux. Một cấu trúc dữ liệu nhân duy nhất tồn tại cho mỗi quá trình trong hệ thống. Một cấu trúc dữ liệu nhân duy nhất tồn tại cho mỗi quá trình trong hệ thống. Tuy nhiên, tốt hơn lưu trữ dữ liệu cho mỗi quá trình trong cấu trúc dữ liệu là nó chứa các con trỏ chỉ tới các cấu trúc dữ liệu khác nơi dữ liệu này được được lưu. Thí dụ, cấu trúc dữ liệu trên quá trình chứa các con trỏ chỉ tới các cấu trúc dữ liệu khác hiện diện danh sách tập tin đang mở, thông tin quản lý tín hiệu, và bộ nhớ ảo. Khi fork được gọi, một quá trình mới được tạo cùng với một bản sao của tất cả cấu trúc dữ liệu của quá trình cha được liên kết tới. Khi lời gọi hệ thống clone được thực hiện, một quá trình mới chỉ tới cấu trúc dữ liệu của quá trình cha, do đó cho phép quá trình con chia sẻ bộ nhớ và tài nguyên của quá trình cha. Một tập hợp cờ được truyền như một tham số tới lời gọi hệ thống clone. Tập hợp cờ này được dùng để hiển thị bao nhiêu quá trình cha được chia sẻ với quá trình con. Nếu không có cờ nào được đặt, không có chia sẻ xảy ra và clone hoạt động giống như fork. Nếu tất cả năm cờ được đặt, quá trình con chia sẻ mọi thứ với quá trình cha. Sự kết hợp khác của cờ cho phép các cấp độ chia sẻ khác nhau giữa hai mức độ cao nhất này.

Điều thú vị là Linux không phân biệt giữa quá trình và luồng. Thật vậy, Linux thường sử dụng thuật ngữ tác vụ-hơn là quá trình hay luồng-khi tham chiếu tới dòng điều khiển trong chương trình. Ngoài quá trình được nhân bản, Linux không hỗ trợ đa luồng, cấu trúc dữ liệu riêng hay thủ

tục nhân. Tuy nhiên, những cài đặt Pthreads là sẳn dùng cho đa luồng cấp người dùng.

Luồng Java

Như chúng ta đã thấy, hỗ trợ cho luồng có thể được cung cấp tại cấp người dùng với một thư viện như Pthread. Hơn nữa, hầu hết hệ điều hành cung cấp sự hỗ trợ cho luồng tại cấp nhân. Java là một trong số nhỏ ngôn ngữ cung cấp sự hỗ trợ tại cấp ngôn ngữ cho việc tạo và quản lý luồng. Tuy nhiên, vì các luồng được quản lý bởi máy ảo Java, không bởi một thư viện cấp người dùng hay nhân, rất khó để phân cấp luồng Java như cấp độ người dùng hay cấp độ nhân. Trong phần này chúng ta trình bày các luồng Java như một thay đổi đối với mô hình người dùng nghiêm ngặt hay mô hình cấp nhân. Phần sau chúng ta sẽ thảo luận một luồng Java có thể được ánh xạ tới luồng nhân bên dưới như thế nào.

Tất cả chương trình tạo ít nhất một luồng điều khiển đơn. Thậm chí một chương trình Java chứa chỉ một phương thức main chạy như một luồng đơn trong máy ảo Java. Ngoài ra, Java cung cấp các lệnh cho phép người phát triển tạo và thao tác các luồng điều khiển bổ sung trong chương trình.

Tạo luồng

Một cách để tạo một luồng rõ ràng là tạo một lớp mới được phát sinh từ lớp Thread và viết đè phương thức run của lớp Thread. Tiếp cận này được hiển thị trong hình sau, ấn bản Java của chương trình đa luồng xác định tổng các số nguyên không âm.

Một đối tượng của lớp phát sinh sẽ chạy như một luồng điều khiển đơn trong máy ảo Java. Tuy nhiên, tạo một đối tượng được phát sinh từ lớp Thread không tạo một luồng mới, trái lại phương thức start mới thật sự tạo luồng mới. Gọi phương thức start cho đối tượng mới thực hiện hai thứ:

1. Nó cấp phát bộ nhớ và khởi tạo một luồng mới trong máy ảo Java.
2. Nó gọi phương thức run, thực hiện luồng thích hợp để được chạy bởi máy ảo Java. (Chú ý, không thể gọi phương thức run trực tiếp, gọi phương thức start sẽ gọi phương thức run)

```
Class Summation extends Thread{public Summation (int n){upper=
n;}public void run(){int sum = 0;if (upper>0){for(int i = 1; i<= upper; i++)
{sum+=i;}System.out.println(“The sum of ”+upper+ “ is “ +
sum);}privateint upper;}public class ThreadTester{public static void
main(String args[]){if(args.length>0){if(Integer.parseInt(args[0])
<0)System.err.println(args[0] + “ must be >= 0.”);else{Summation thrd =
new Summation
(Integer.parseInt(args[0]));Thrd.start();}}ElseSystem.out.println(“Usage:
summation < integer value”);}}
```

Hình IV-8- Chương trình Java để tính tổng số nguyên không âm

Khi chương trình tính tổng thực thi, hai luồng được tạo bởi JVM. Luồng đầu tiên là luồng được nối kết với ứng dụng-luồng này bắt đầu thực thi tại phương thức main. Luồng thứ hai là luồng Summation được tạo rõ ràng với phương thức start. Luồng Summation bắt đầu thực thi trong phương thức run của nó. Luồng kết thúc khi nó thoát khỏi phương thức run của nó.

JVM và hệ điều hành chủ

Cài đặt điển hình của JVM ở trên đỉnh của hệ điều hành chủ (host operating system). Thiết lập này cho phép JVM che giấu chi tiết cài đặt của hệ điều hành bên dưới và cung cấp môi trường không đổi, trừu tượng cho phép chương trình Java hoạt động trên bất kỳ phần cứng nào hỗ trợ JVM. Đặc tả cho JVM không hiển thị các luồng Java được ánh xạ tới hệ điều hành bên dưới như thế nào để thay thế việc quên đi việc quyết định cài đặt cụ thể của JVM. Windows 95/98/NT và Windows 2000 dùng mô hình một-một; do đó, mỗi luồng Java cho một JVM chạy trên các hệ điều hành này ánh xạ tới một luồng nhân. Solaris 2 khởi đầu cài

đặt JVM dùng mô hình nhiều-một. Tuy nhiên, ấn bản 1.1 của JVM với Solaris 2.6 được cài đặt dùng mô hình nhiều-nhiều.

Tóm tắt

Luồng là một dòng điều khiển trong phạm vi một quá trình. Quá trình đa luồng gồm nhiều dòng điều khiển khác nhau trong cùng không gian địa chỉ. Những lợi điểm của đa luồng gồm đáp ứng nhanh đối với người dùng, chia sẻ tài nguyên trong quá trình, tính kinh tế, và khả năng thuận lợi trong kiến trúc đa xử lý.

Luồng cấp người dùng là các luồng được nhìn thấy bởi người lập trình và không được biết bởi nhân. Thư viện luồng trong không gian người dùng điển hình quản lý luồng cấp người dùng. Nhân của hệ điều hành hỗ trợ và quản lý các luồng cấp nhân. Thông thường, luồng cấp người dùng nhanh hơn luồng cấp nhân trong việc tạo và quản lý. Có ba loại mô hình khác nhau liên quan đến luồng cấp người dùng và luồng cấp nhân. Mô hình nhiều-một ánh xạ nhiều luồng người dùng tới một luồng nhân. Mô hình một-một ánh xạ mỗi luồng người dùng tới một luồng nhân tương ứng. Mô hình nhiều-nhiều đa hợp nhiều luồng người dùng tới một số lượng nhỏ hơn hay bằng luồng nhân.

Những chương trình đa luồng giới thiệu nhiều thử thách cho việc lập trình, gồm ngữ nghĩa của lời gọi hệ thống fork và exec. Những vấn đề khác gồm huỷ bỏ luồng, quản lý tín hiệu, và dữ liệu đặc tả luồng. Nhiều hệ điều hành hiện đại cung cấp nhân hỗ trợ luồng như Windows NT, Windows 2000, Solaris 2 và Linux. Pthread API cung cấp tập hợp các hàm để tạo và quản lý luồng tại cấp người dùng. Java cung cấp một API tương tự cho việc hỗ trợ luồng. Tuy nhiên, vì các luồng Java được quản lý bởi JVM và không phải thư viện luồng cấp người dùng hay nhân, chúng không rơi vào loại luồng người dùng hay nhân.

Đồng bộ hóa quá trình
1 Mục tiêu Sau khi học xong chương này, người học nắm được những kiến thức sau: - Hiểu vấn đề vùng tương trục - Hiểu cơ chế hoạt động hiệu báo Semaphores để đồng bộ hóa quá trình - Hiểu cơ chế hoạt động của Monitors để đồng bộ hóa quá trình - Vận dụng các giải pháp để giải quyết các bài toán đồng bộ hóa cơ bản

Giới thiệu

Một quá trình hợp tác là một quá trình có thể gây ảnh hưởng hay bị ảnh hưởng tới quá trình khác đang thực thi trong hệ thống. Các quá trình hợp tác có thể chia sẻ trực tiếp không gian địa chỉ luận lý (mã và dữ liệu), hay được phép chia sẻ dữ liệu thông qua các tập tin. Trường hợp đầu đạt được thông qua việc sử dụng các quá trình có trọng lượng nhẹ hay luồng. Truy xuất đồng hành dữ liệu được chia sẻ có thể dẫn tới việc không đồng nhất dữ liệu. Trong chương này chúng ta sẽ thảo luận các cơ chế đảm bảo việc thực thi có thứ tự của các quá trình hợp tác chia sẻ không gian địa chỉ để tính đúng đắn của dữ liệu luôn được duy trì.

Tổng quan

Trong chương trước, chúng ta phát triển một mô hình hệ thống chứa số lượng quá trình hợp tác tuần tự, tất cả chúng chạy bất đồng bộ và có thể chia sẻ dữ liệu. Chúng ta hiển thị mô hình này với cơ chế vùng đệm có kích thước giới hạn, được đại diện cho hệ điều hành.

Chúng ta xét giải pháp bộ nhớ được chia sẻ cho bài toán vùng đệm có kích thước giới hạn. Giải pháp này cho phép có nhiều nhất BUFFER_SIZE –1 sản phẩm trong vùng đệm tại cùng thời điểm. Giả sử rằng chúng ta muốn hiệu chỉnh giải thuật để giải quyết sự thiếu sót này. Một khả năng là thêm một biến đếm số nguyên counter, được khởi tạo bằng 0. counter được tăng mỗi khi chúng ta thêm một sản phẩm tới vùng đệm và bị giảm mỗi khi chúng ta lấy một sản phẩm ra khỏi vùng đệm. Mã cho quá trình người sản xuất có thể được hiệu chỉnh như sau:

while (1){/*tạo sản phẩm trong nextProduced*/while (counter==BUFFER_SIZE); /*không làm gì cả*/buffer[in] = nextProduced;in = (in + 1) % BUFFER_SIZE;counter++;}

Mã cho quá trình người tiêu dùng có thể được hiệu chỉnh như sau:

while (1){while (counter == 0) ; /*không làm gì cả*/nextConsumed = buffer[out];out = (out + 1) % BUFFER_SIZE;counter--;/*tiêu thụ sản phẩm trong nextConsumed*/}

Mặc dù cả hai thủ tục người sản xuất và người tiêu dùng thực thi đúng khi tách biệt nhau nhưng chúng không thực hiện đúng chức năng khi thực thi đồng hành. Như minh hoạ dưới đây, giả sử rằng giá trị của biến counter hiện tại là 5 và thủ tục người sản xuất và người tiêu dùng thực thi đồng hành câu lệnh "counter++" và "counter--". Theo sau việc thực thi hai câu lệnh này, giá trị của biến counter có thể là 4, 5 hay 6! Kết quả chỉ đúng khi biến counter==5, được tạo ra đúng nếu quá trình người sản xuất và người tiêu dùng thực thi riêng biệt.

Chúng ta có thể minh hoạ giá trị của counter có thể không đúng như sau. Chú ý, câu lệnh "counter++" có thể được cài đặt bằng ngôn ngữ máy (trên một máy điển hình) như sau:

register1 = counter

register1 = register1 + 1

counter = register1

Ở đây register1 là một thanh ghi CPU cục bộ. Tương tự, câu lệnh "counter--" được cài đặt như sau:

register2 = counter

register2 = register2 - 1

counter = register2

Ở đây register2 là thanh ghi CPU cục bộ. Dù là register1 và register2 có thể dùng cùng thanh ghi vật lý, nhưng nội dung của thanh ghi sẽ được lưu lại và lấy lại bởi bộ quản lý ngắt.

Thực thi đồng hành của “counter++” và “counter--” là tương tự như thực thi tuần tự ở đây các câu lệnh cấp thấp hơn được hiện diện trước bị phủ lắp trong thứ tự bất kỳ (nhưng thứ tự bên trong mỗi câu lệnh cấp cao được lưu giữ). Một sự phủ lắp là:

T0: producerthực thiregister1 = counter{register1 = 5}

T1: producerthực thiregister1 = register1 + 1 {register1 = 6}

T2: consumerthực thiregister2 = counter{register2 = 5}

T3: consumerthực thiregister2 = register2 – 1{register2 = 4}

T4: producerthực thicounter = register1{counter = 6}

T5: consumerthực thicounter = register2{counter = 4}

Chú ý rằng, chúng ta xem xét tình trạng không đúng “counter==4” theo đó có 4 vùng đệm đầy, nhưng thực tế khi đó có 5 vùng đệm đầy. Nếu chúng đổi ngược lại thứ tự của câu lệnh T4 và T5, chúng ta sẽ có trạng thái không đúng “counter ==6”.

Chúng ta đi đến trạng thái không đúng này vì chúng ta cho phép cả hai quá trình thao tác đồng thời trên biến counter. Trường hợp tương tự, ở đây nhiều quá trình truy xuất và thao tác cùng dữ liệu đồng hành và kết quả của việc thực thi phụ thuộc vào thứ tự xác định trong đó việc truy xuất xảy ra, được gọi là điều kiện cạnh tranh (race condition). Để ngăn chặn điều kiện cạnh tranh ở trên, chúng ta cần đảm bảo rằng chỉ một quá trình tại một thời điểm có thể được thao tác biến counter. Để thực hiện việc đảm bảo như thế, chúng ta yêu cầu một vài hình thức đồng bộ hoá quá trình. Những trường hợp như thế xảy ra thường xuyên trong các hệ điều hành khi các phần khác nhau của hệ thống thao tác các tài nguyên và chúng ta muốn các thay đổi không gây trở ngại một sự thay đổi khác.

Phần chính của chương này là tập trung vào vấn đề đồng bộ hoá và cộng tác quá trình.

Vấn đề vùng tương trục

Xét một hệ thống gồm n quá trình (P0, P1, … ,Pn-1). Mỗi quá trình có một phân đoạn mã, được gọi là vùng tương trục (critical section), trong đó quá trình này có thể thay đổi những biến dùng chung, cập nhật một bảng, viết đến tập tin,.. Đặc điểm quan trọng của hệ thống là ở chỗ, khi một quá trình đang thực thi trong vùng tương trục, không có quá trình nào khác được phép thực thi trong vùng tương trục của nó. Do đó, việc thực thi của các vùng tương trục bởi các quá trình là sự loại trừ hỗ tương. Vấn đề vùng tương trục là thiết kế một giao thức mà các quá trình có thể dùng để cộng tác. Mỗi quá trình phải yêu cầu quyền để đi vào vùng tương trục của nó. Vùng mã thực hiện yêu cầu này là phần đi vào (entry section). Vùng tương trục có thể được theo sau bởi một phần kết thúc (exit section). Mã còn lại là phần còn lại (remainder section).

entry sectiondo {critical sectionexit sectionremainder section}while (1);
Hình V-1 Cấu trúc chung của một quá trình điển hình Pi

Một giải pháp đối với vấn đề vùng tương trục phải thoả mãn ba yêu cầu sau:

1. Loại trừ hỗ tương (Mutual Exclusion): Nếu quá trình Pi đang thực thi trong vùng tương trục của nó thì không quá trình nào khác đang được thực thi trong vùng tương trục đó.
2. Tiến trình (Progress): nếu không có quá trình nào đang thực thi trong vùng tương trục và có vài quá trình muốn vào vùng tương trục thì chỉ

những quá trình không đang thực thi phần còn lại mới có thể tham gia vào việc quyết định quá trình nào sẽ đi vào vùng tương trục tiếp theo và chọn lựa này không thể trì hoãn vô hạn định.

3. Chờ đợi có giới hạn (bounded wait): giới hạn số lần các quá trình khác được phép đi vào miền tương trục sau khi một quá trình thực hiện yêu cầu để đi vào miền tương trục của nó và trước khi yêu cầu đó được gán.

Chúng ta giả sử rằng mỗi quá trình đang thực thi với tốc độ khác 0. Tuy nhiên, chúng ta có thể thực hiện rằng không có giả thuyết nào được quan tâm về tốc tương đối của n quá trình.

Trong phần tiếp theo chúng ta nghiên cứu để nắm được các giải pháp thoả ba yêu cầu này. Những giải pháp này không quan tâm đến các chỉ thị phần cứng hay số lượng bộ xử lý mà phần cứng hỗ trợ. Tuy nhiên chúng ta giả sử rằng những chỉ thị ngôn ngữ máy cơ bản (chỉ thị cơ bản như load, store và test) được thực hiện mang tính nguyên tử (atomically). Nghĩa là, nếu hai chỉ thị như thế được thực thi đồng hành thì kết quả tương tự như thực thi tuần tự trong thứ tự không xác định. Do đó, nếu chỉ thị load và store được thực thi đồng hành thì load sẽ nhận giá trị cũ hay mới như không có sự kết hợp vừa cũ vừa mới.

Khi trình bày một giải thuật, chúng ta định nghĩa chỉ những biến được dùng cho mục đích đồng bộ và mô tả chỉ một quá trình điển hình Pi mà cấu trúc của nó được hiển thị trong hình V.1. Phần đi vào và kết thúc được bao trong hình chữ nhật để nhấn mạnh các đoạn mã quan trọng.

while (turn!=i) ;do {critical sectionturn = j;remainder section}while (1);

Hình V-2-Cấu trúc của quá trình Pi trong giải thuật 1

Giải pháp

Có nhiều giải pháp để thực hiện việc loại trừ hỗ tương. Các giải pháp này, tuỳ thuộc vào cách tiếp cận trong xử lý của quá trình bị khoá, được

phân biệt thành hai lớp: chờ đợi bận (busy waiting) và nghẽn và đánh thức (sleep and wakeup)

Giải pháp “chờ đợi bận”

Giải pháp hai quá trình (two-Process Solution)

Trong phần này, chúng ta giới hạn việc quan tâm tới những giải thuật có thể áp dụng chỉ hai quá trình cùng một lúc. Những quá trình này được đánh số P0 và P1. Để thuận lợi, khi trình bày Pi, chúng ta dùng Pj để chỉ quá trình còn lại, nghĩa là j = 1 – i

Giải thuật 1

Tiếp cận đầu tiên của chúng ta là để hai quá trình chia sẻ một biến số nguyên chung turn được khởi tạo bằng 0 (hay 1). Nếu turn == 0 thì quá trình Pi được phép thực thi trong vùng tương trục của nó. Cấu trúc của quá trình Pi được hiển thị trong Hình V.-2.

Giải pháp này đảm bảo rằng chỉ một quá trình tại một thời điểm có thể ở trong vùng tương trục của nó. Tuy nhiên, nó không thoả mãn yêu cầu tiến trình vì nó yêu cầu sự thay đổi nghiêm khắc của các quá trình trong việc thực thi của vùng tương trục. Thí dụ, nếu turn == 0 và P1 sẵn sàng đi vào vùng tương trục của nó thì P1 không thể đi vào vùng tương trục thậm chí khi P0 đang ở trong phần còn lại của nó.

Giải thuật 2

Vấn đề với giải thuật 1 là nó không giữ lại đủ thông tin về trạng thái của mỗi quá trình; nó nhớ chỉ quá trình nào được phép đi vào miền tương trục. Để giải quyết vấn đề này, chúng ta có thể thay thế biến turn với mảng sau:

Boolean flag[2];

Các phần tử của mảng được khởi tạo tới flase. Nếu flag[i] là true, giá trị này hiển thị rằng Pi sẵn sàng đi vào vùng tương trục. Cấu trúc của quá trình Pi được hiển thị trong hình V.-3 dưới đây:

flag[i] = true;while (flag[j]);do{critical sectionflag[i] = false;remainder section} while(1);

Hình V-3 –Cấu trúc của quá trình Pi trong giải thuật 2

Trong giải thuật này, quá trình Pi trước tiên thiết lập flag[i] tới true, hiển thị rằng nó sẵn sàng đi vào miền tương trục. Sau đó, Pi kiểm tra rằng quá trình quá trình Pj cũng không sẵn sàng đi vào miền tương trục của nó. Nếu Pj sẵn sàng thì Pi sẽ chờ cho tới khi Pj hiển thị rằng nó không còn cần ở trong vùng tương trục nữa (nghĩa là cho tới khi flag[j] là false). Tại thời điểm này, Pi sẽ đi vào miền tương trục. Thoát ra khỏi miền tương trục, Pi sẽ đặt flag[i] là false, cho phép quá trình khác (nếu nó đang chờ) đi vào miền tương trục của nó.

Trong giải pháp này, yêu cầu loại trừ hỗ tương sẽ được thoả mãn. Tuy nhiên, yêu cầu tiến trình không được thoả mãn. Để minh hoạ vấn đề này, chúng ta xem xét thứ tự thực thi sau:

T0: P0 thiết lập flag[0] = true;

T1: P1 thiết lập flag[1] = true;

Bây giờ P0 và P1 được lập mãi mãi trong câu lệnh while tương ứng của chúng.

Giải thuật này phụ thuộc chủ yếu vào thời gian chính xác của hai quá trình. Thứ tự này được phát sinh trong môi trường nơi có nhiều bộ xử lý thực thi đồng hành hay nơi một ngắt (chẳng hạn như một ngắt định thời) xảy ra lập tức sau khi bước T0 được thực thi và CPU được chuyển từ một quá trình này tới một quá trình khác.

Chú ý rằng chuyển đổi thứ tự của các chỉ thị lệnh để thiết lập flag[i] và kiểm tra giá trị của flag[j] sẽ không giải quyết vấn đề của chúng ta. Hơn nữa chúng ta sẽ có một trường hợp đó là hai quá trình ở trong vùng tương trục cùng một lúc, vi phạm yêu cầu loại trừ hỗ tương.

Giải thuật 3

Giải thuật 3 còn gọi là giải pháp Peterson. Bằng cách kết hợp hai ý tưởng quan trọng trong giải thuật 1 và 2, chúng ta đạt được một giải pháp đúng tới với vấn đề vùng tương trục, ở đó hai yêu cầu được thoả. Các quá trình chia sẻ hai biến:

Boolean flag[2]

Int turn;

Khởi tạo flag[0] = flag[1] = false và giá trị của turn là không xác định (hoặc là 0 hay 1). Cấu trúc của quá trình Pi được hiển thị trong hình sau:

do{flag[i] = true;turn = j;while (flag[j] &&turn ==j);critical sectionflag[i] = false;remainder section} while (1);

Hình V-4 Cấu trúc của quá trình Pi trong giải thuật 3

Để đi vào miền tương trục, quá trình Pi trước tiên đặt flag[i] là true sau đó đặt turn tới giá trị j, do đó xác định rằng nếu quá trình khác muốn đi vào miền tương trục nó. Nếu cả hai quá trình đi vào miền tương trục cùng một lúc turn sẽ đặt cả hai i và j tại xấp xỉ cùng một thời điểm. Chỉ một trong hai phép gán này là kết quả cuối cùng. Giá trị cuối cùng của turn quyết định quá trình nào trong hai quá trình được cho phép đi vào miền tương trục trước.

Bây giờ chúng ta chứng minh rằng giải pháp này là đúng. Chúng ta cần hiển thị rằng:

1. Loại trừ hỗ tương được bảo toàn

2. Yêu cầu tiến trình được thoả
3. Yêu cầu chờ đợi có giới hạn cũng được thoả

Chứng minh thuộc tính 1, chúng ta chú ý rằng mỗi Pi đi vào miền tương trục của nó chỉ nếu flag[j] ==false hay turn ==i. Cũng chú ý rằng, nếu cả hai quá trình có thể đang thực thi trong vùng tương trục của chúng tại cùng thời điểm thì flag[0] == flag[1] ==true. Hai nhận xét này ngụ ý rằng P0 và P1 không thể thực thi thành công trong vòng lặp while của chúng tại cùng một thời điểm vì giá trị turn có thể là 0 hay 1. Do đó, một trong các quá trình-Pj phải được thực thi thành công câu lệnh while, ngược lại Pi phải thực thi ít nhất câu lệnh bổ sung ("turn==j"). Tuy nhiên, vì tại thời điểm đó, flag[j] ==true và turn ==j, và điều kiện này sẽ không đổi với điều kiện là Pj ở trong vùng miền tương trục của nó, kết quả sau việc loại trừ hỗ tương được bảo vệ

do {		
	flag[i] = true;turn = j;while (flag[j] && turn ==j);	
	critical section	
	flag[i] = false;	
	Remainder section	
}while (1);		
Hình V-5-Cấu trúc của quá trình Pi trong giải thuật 3		

Để chứng minh thuộc tính 2 và 3, chúng ta chú ý rằng một quá trình Pi có thể được ngăn chặn từ việc đi vào miền tương truc chỉ nếu nó bị kẹt trong vòng lặp while với điều kiện flag[j] == true và turn == j. Nếu Pj không sẵn sàng đi vào miền tương trục thì flag[j] == false và Pi có thể đi vào miền tương trục của nó. Nếu Pj đặt flag[j] là true và nó cũng đang thực thi trong câu lệnh while của nó thì turn == i hay turn == j. Nếu turn == i thì Pi sẽ đi vào miền tương trục. Nếu turn ==j thì Pj sẽ đi vào miền tương trục. Tuy nhiên, một khi Pj ở trong vùng tương trục của nó thì nó sẽ đặt lại flag[j] tới false, cho phép Pi đi vào miền tương trục của nó. Nếu Pj đặt lại flag[j] tới true, nó cũng phải đặt turn tới i. Do đó, vì Pi không thay đổi giá trị của biến turn trong khi thực thi câu lệnh while, nên Pi sẽ đi vào miền tương trục (tiến trình) sau khi nhiều nhất chỉ Pj đi vào (chờ có giới hạn).

Giải pháp nhiều quá trình

Giải thuật 3 giải quyết vấn đề miền tương trục cho hai quá trình. Bây giờ chúng ta phát triển một giải thuật để giải quyết vấn đề miền tương trục cho n quá trình. Giải thuật này được gọi là giải thuật Bakery và nó dựa trên cơ sở của giải thuật định thời thường được dùng trong cửa hiệu bánh mì, cửa hàng kem,..nơi mà thứ tự rất hỗn độn. Giải thuật này được phát triển cho môi trường phân tán, nhưng tại thời điểm này chúng ta tập trung chỉ những khía cạnh của giải thuật liên quan tới môi trường tập trung.

Đi vào một cửa hàng, mỗi khách hàng nhận một số. Khách hàng với số thấp nhất được phục vụ tiếp theo. Tuy nhiên, giải thuật Bakery không thể đảm bảo hai quá trình (khách hàng) không nhận cùng số. Trong trường hợp ràng buộc, một quá trình với tên thấp được phục vụ trước. Nghĩa là, nếu Pi và Pj nhận cùng một số và nếu (i < j) thì Pi được phục vụ trước. Vì tên quá trình là duy nhất và được xếp thứ tự nên giải thuật là hoàn toàn mang tính “may rủi” (deterministic).

Cấu trúc dữ liệu chung là

booleanchoosing[n];

intnumber[n];

Đầu tiên, các cấu trúc dữ liệu này được khởi tạo tới false và 0 tương ứng. Để tiện dụng, chúng ta định nghĩa các ký hiệu sau:

- (a, b) < (c, d) nếu a< c hay nếu a==c và b< d
- max(a0,…,an-1) là số k   ai với i = 0,…,n-1

Cấu trúc của quá trình Pi được dùng trong giải thuật Bakery, được hiển thị trong hình dưới đây.

do {		
	choosing[i] = true;number[i] = max(number[0], number[i],…,number[n-1]) + 1;choosing[i] = false;for (j=0; j < n; j++){while (choosing[j]);while ((number[j]!=0)&&((number[j], j) <(number[i], i)));}	
	Critical section	
	Number[i] = 0;	
}	While (1);	

Hình V-6 Cấu trúc của giải thuật Pi trong giải thuật Bakery

Kết quả được cho này thể hiện rằng loại trừ hỗ tương được tuân theo. Thật vậy, xét Pi trong vùng tương trục của nó và Pk cố gắng đi vào vùng tương trục Pk. Khi quá trình Pk thực thi câu lệnh while thứ hai cho j==i, nhận thấy rằng

- number[i] != 0

- (number[i], i) < (number[k], k).

Do đó, nó tiếp tục vòng lặp trong câu lệnh while cho đến khi Pi rời khỏi vùng tương trục Pi.

Giải thuật trên đảm bảo rằng yêu cầu về tiến trình, chờ đợi có giới hạn và đảm bảo sự công bằng, vì các quá trình đi vào miền tương trục dựa trên cơ sở tới trước được phục vụ trước.

Phần cứng đồng bộ hoá

Như các khía cạnh khác của phần mềm, các đặc điểm phần cứng có thể làm các tác vụ lập trình dễ hơn và cải tiến tính hiệu quả của hệ thống. Trong phần này, chúng ta trình bày một số chỉ thị phần cứng đơn giản sẵn dùng trên nhiều hệ thống và trình bày cách chúng được dùng hiệu quả trong việc giải quyết vấn đề miền tương trục.

```
boolean TestAndSet( boolean &target){boolean rv = target;target =
true;returnrv;}
```

Hình V-7 Định nghĩa của chỉ thị TestAndSet

Vấn đề miền tương trục có thể được giải quyết đơn giản trong môi trường chỉ có một bộ xử lý nếu chúng ta cấm các ngắt xảy ra khi một biến chia sẻ đang được thay đổi. Trong cách này, chúng ta đảm bảo rằng chuỗi chỉ thị hiện hành có thể được cho phép thực thi trong thứ tự không trưng dụng. Không có chỉ thị nào khác có thể chạy vì thế không có bất cứ sự thay đổi nào có thể được thực hiện trên các biến được chia sẻ.

Tuy nhiên, giải pháp này là không khả thi trong một môi trường có nhiều bộ xử lý. Vô hiệu hoá các ngắt trên đa bộ xử lý có thể mất nhiều thời gian khi một thông điệp muốn truyền qua tất cả bộ xử lý. Việc truyền thông điệp này bị trì hoãn khi đi vào miền tương trục và tính hiệu quả của hệ thống bị giảm.

Do đó nhiều máy cung cấp các chỉ thị phần cứng cho phép chúng ta kiểm tra hay thay đổi nội dung của một từ (word) hay để thay đổi nội dung của

hai từ tuân theo tính nguyên tử (atomically)-như là một đơn vị không thể ngắt. Chúng ta có thể sử dụng các chỉ thị đặc biệt này để giải quyết vấn đề miền tương trục trong một cách tương đối đơn giản.

Chỉ thị TestAndSet có thể được định nghĩa như trong hình V.-7. Đặc điểm quan trọng của chỉ thị này là việc thực thi có tính nguyên tử. Do đó, nếu hai chỉ thị TestAndSet được thực thi cùng một lúc (mỗi chỉ thị trên một CPU khác nhau), thì chúng sẽ được thực thi tuần tự trong thứ tự bất kỳ.

do{	
	while (TestAndSet(lock));
	Critical section
	lock:= false
	remainder section
} while (1);	

Hình V-8: Cài đặt loại trừ hỗ tương với TestAndSet

Nếu một máy hỗ trợ chỉ thị TestAndSet thì chúng ta có thể loại trừ hỗ tương bằng cách khai báo một biến khoá kiểu luận lý và được khởi tạo tới false. Cấu trúc của quá trình Pi được hiển thị trong hình V.-9 ở trên.

Chỉ thị Swap được định như hình V.-9 dưới đây, thao tác trên nội dung của hai từ; như chỉ thị TestAndSet, nó được thực thi theo tính nguyên tử.

void Swap(boolean &a, boolean &b){

boolean temp = a;

a = b;

b = temp;

}

Hình V-9: Định nghĩa chỉ thị Swap

Nếu một máy hỗ trợ chỉ thị Swap, thì việc loại trừ hỗ tương có thể được cung cấp như sau. Một biến luận lý toàn cục lock được khai báo và được khởi tạo tới false. Ngoài ra, mỗi quá trình cũng có một biến luận lý cục bộ key. Cấu trúc của quá trình Pi được hiển thị trong hình V.-10 dưới đây.

do{		
	key = true;while (key == true) Swap(lock, key);	
	Critical section	
	lock = false;	
	Remainder section	
} while(1);		

Hình V-10: Cài đặt loại trừ hỗ tương với chỉ thị Swap

Các giải thuật này không thoả mãn yêu cầu chờ đợi có giới hạn. Chúng ta hiển thị giải thuật sử dụng chỉ thị TestAndSet trong hình V.-11 dưới đây. Giải thuật này thoả mãn tất cả các yêu cầu miền tương trục.

do{	
	Waiting[i] = true;key = true;while (waiting[i] && key) key = TestAndSet(lock);waiting[i] = false;
	Critical section
	j = (i + 1) % n;while ((j != i) && !waiting[j])j = (j + 1) % n;if (j == i) lock = false;elsewaiting[j] = false;
	Remainder section
} while(1);	

Hình V-11 Loại trừ hỗ tương chờ đợi có giới hạn với TestAndSet

Cấu trúc dữ liệu thông thường là:

boolean waiting[n];

booleanlock;

Cấu trúc dữ liệu này được khởi tạo tới false. Để chứng minh rằng loại trừ hỗ tương được thoả, chúng ta chú ý rằng quá trình Pi có thể đưa vào miền tương trục chỉ nếu hoặc waiting[i] ==false hay key == false. Giá trị của key có thể trở thành false chỉ nếu TestAndSet được thực thi. Đối với quá trình đầu tiên, để thực thi TestAndSet sẽ tìm key == false; tất cả quá trình khác phải chờ. Biến waiting[i] có thể trở thành false chỉ nếu quá trình khác rời khởi miền tương trục của nó; chỉ một waiting[i] được đặt false, duy trì yêu cầu loại trừ hỗ tương.

Để chứng minh yêu cầu tiến trình được thoả, chúng ta chú ý rằng các đối số được hiện diện cho việc loại trừ hỗ tương cũng áp dụng được ở đây, vì thế một quá trình thoát khỏi miền tương trục hoặc đặt lock bằng false

hay đặt waiting[j] bằng false. Cả hai trường hợp đều cho phép một quá trình đang chờ để đi vào miền tương trục được xử lý.

Để chứng minh yêu cầu chờ đợi được giới hạn được thoả, chúng ta chú ý rằng khi một quá trình rời miền tương trục của nó, nó duyệt qua mảng waiting trong thứ tự tuần hoàn (i + 1, i + 2, …, n – 1, 0, …, i - 1). Nó định rõ quá trình đầu tiên trong thứ tự này mà thứ tự đó ở trong phần đi vào (waiting[j] == true) khi quá trình tiếp theo đi vào miền tương trục. Bất cứ quá trình nào đang chờ để đi vào miền tương trục sẽ thực hiện n – 1 lần. Tuy nhiên, đối với người thiết kế phần cứng, cài đặt các chỉ thị nguyên tử TestAndSet trên bộ đa xử lý không là tác vụ thử nghiệm.

Những giải pháp trên đều phải thực hiện một vòng lặp để kiểm tra liệu nó có được phép vào miền tương trục hay không. Nếu điều kiện chưa thoả, quá trình phải chờ tiếp tục trong vòng lặp kiểm tra này. Các giải pháp buộc quá trình phải liên tục kiểm tra điều kiện để phát hiện thời điểm thích hợp được vào miền tương trục như thế được gọi là các giải pháp chờ đợi bận “busy waiting”. Lưu ý, việc kiểm tra như thế tiêu thụ rất nhiều thời gian sử dụng CPU, do vậy quá trình đang chờ vẫn chiếm dụng CPU. Xu hướng giải quyết vấn đề đồng bộ hoá là nên tránh các giải pháp chờ đợi bận.

Các giải pháp “SLEEP and WAKEUP”

Để loại bỏ các bất tiện của của giải pháp chờ đợi bận, chúng ta có thể tiếp cận theo hướng cho một quá trình chưa đủ điều kiện vào miền tương trục chuyển sang trạng thái nghẽn, từ bỏ quyền sử dụng CPU. Để thực hiện điều này, cần phải sử dụng các thủ tục do hệ điều hành cung cấp để thay đổi trạng thái quá trình. Hai thủ tục cơ bản SLEEP và WAKEUP thường được sử dụng cho mục đích này.

SLEEP là một lời gọi hệ thống có tác dụng làm “nghẽn” (blocked) hoạt động của quá trình gọi nó và chờ đến khi được một tiến trình khác “đánh thức”. Lời gọi hệ thống WAKEUP nhận một tham số duy nhất: quá trình sẽ được kích hoạt trở lại (đặt về trạng thái sẵn sàng).

Ý tưởng sử dụng SLEEP và WAKEUP như sau: khi một quá trình chưa đủ điều kiện vào miền tương trục, nó gọi SLEEP để tự khoá đến khi có một quá trình khác gọi WAKEUP để giải phóng nó. Một quá trình gọi WAKEUP khi ra khỏi miền tương trục để đánh thức một quá trình đang chờ, tạo cơ hội cho quá trình này vào miền tương trục.

	int busy;// 1 nếu miền tương trục đang bị chiếmintblocked;// đếm số lượng quá trình đang bị khoádo{	
	if (busy) {blocked = blocked + 1;sleep();}elsebusy = 1;}while (1);	
	Critical section	
	busy= 0;if (blocked){wakeup(process);blocked = blocked -1;}	
	Remainder section	
	Hình V-12 Cấu trúc chương trình trong giải pháp SLEEP and WAKEUP	

Khi sử dụng SLEEP và WAKEUP cần hết sức cẩn thận, nếu không muốn xảy ra tình trạng mâu thuẩn truy xuất trong một vài tình huống như sau: giả sử quá trình A vào miền tương trục, và trước khi nó rời miền tương trục thì quá trình B được kích hoạt. Quá trình B thử vào miền tương trục nhưng nó nhận thấy A đang ở trong đó, do vậy B tăng giá trị biến blocked lên 1 và chuẩn bị gọi SLEEP để tự nghẽn. Tuy nhiên, trước khi B có thể thực hiện SLEEP, quá trình A được kích hoạt trở lại và ra khỏi miền tương trục. Khi ra khỏi miền tương trục, quá trình A nhận

thấy có một quá trình đang chờ (blocked=1) nên gọi WAKEUP và giảm giá trị blocked xuống 1. Khi đó tín hiệu WAKEUP sẽ lạc mất do quá trình B chưa thật sự “ngủ” để nhận tín hiệu đánh thức! Khi quá trình B được tiếp tục xử lý, nó mới gọi SLEEP và tự nghẽn vĩnh viễn!

Vấn đề ghi nhận được là tình trạng lỗi này xảy ra do việc kiểm tra trạng thái miền tương trục và việc gọi SLEEP hay WAKEUP là những hành động tách biệt, có thể bị ngắt nửa chừng trong quá trình xử lý, do đó có khi tín hiệu WAKEUP gởi đến một quá trình chưa bị nghẽn sẽ lạc mất. Để tránh những tình huống tương tự, hệ điều hành cung cấp những cơ chế đồng bộ hoá dựa trên ý tưởng của chiến lược “SLEEP and WAKEUP” nhưng chưa được xây dựng bao gồm cả phương tiện kiểm tra điều kiện vào miền tương trục giúp sử dụng an toàn.

Semaphore

Tiếp cận Semaphore được. Dijkstra đề xuất vào năm 1965. Một semaphore S là một biến số nguyên (integer) được truy xuất chỉ thông qua hai thao tác nguyên tử: wait và signal. Các thao tác này được đặt tên P (cho wait - chờ để kiểm tra) và V (cho signal- báo hiệu để tăng). Định nghĩa cơ bản của wait trong mã giả là:

```
wait(S){

while (S  0)

;//no-op

S--;

}
```

Định nghĩa cơ bản của signal trong mã giả là

```
signal(S){

S++;
```

}

Những sửa đổi đối với giá trị integer của semaphore trong các thao tác wait và signal phải được thực thi không bị phân chia. Nghĩa là khi một quá trình sửa đổi giá trị semaphore, không có quá trình nào cùng một lúc có thể sửa đổi cùng biến semaphore đó. Ngoài ra, trong trường hợp của biến wait(S), kiểm tra giá trị integer của S (S   0) và sửa đổi có thể của nó (S--) cũng phải được thực thi mà không bị ngắt.

Cách dùng

Chúng ta có thể sử dụng semaphores để giải quyết vấn đề miền tương trục với n quá trình. N quá trình chia sẻ một biến semaphore, mutex (viết tắt từ mutual exclusion) được khởi tạo 1. Mỗi quá trình Pi được tổ chức như được hiển thị trong hình dưới đây.

do{		
	Wait(mutex)	
	critical section	
	Signal(mutex)	
	remainder section	
}while(1);		

Hình V-13 Cài đặt loại trừ hỗ tương với semaphores

Chúng ta cũng sử dụng semaphores để giải quyết các vấn đề đồng bộ khác nhau. Thí dụ, để xem xét hai quá trình đang thực thi đồng hành: P1 với câu lệnh S1 và P2 với câu lệnh S2. Giả sử chúng ta yêu cầu rằng S2 được thực thi chỉ sau khi S1 hoàn thành. Chúng ta có thể hoàn thành cơ chế này một cách dễ dàng bằng cách P1 và P2 chia sẻ một semaphore chung synch, được khởi tạo 0 và bằng cách chèn các câu lệnh:

S1;

signal(sync);

vào quá trình P1 và các câu lệnh:

wait(synch);

S2;

vào trong quá trình P2. Vì synch được khởi tạo 0. P2 sẽ thực thi S2 chỉ sau khi P1 nạp signal(synch) mà sau đó là S1;

Cài đặt

Nhược điểm chính của các giải pháp loại trừ hỗ tương trong phần V.-5.1 và của semaphore được cho ở đây là tất cả chúng đều đòi hỏi sự chờ đợi bận.Để giải quyết yêu cầu cho việc chờ đợi bận, chúng ta có thể hiệu chỉnh định nghĩa của các thao tác wait và signal của semaphore. Khi một quá trình thực thi thao tác wait và nhận thấy rằng nếu giá trị của semaphore không dương, nó phải chờ. Tuy nhiên, thay vì chờ đợi bận, quá trình có thể nghẽn chính nó. Thao tác nghẽn đặt quá trình vào một hàng đợi gắn liền với semaphore và trạng thái quá trình được chuyển tới trạng thái chờ. Sau đó, điều khiển được chuyển tới bộ định thời biểu và bộ định thời biểu chọn một quá trình khác để thực thi.

Một quá trình bị nghẽn chờ trên biến semaphore nên được khởi động lại khi quá trình khác thực thi thao tác signal. Quá trình được khởi động lại bởi thao tác wakeup và chuyển quá trình từ trạng thái chờ sang trạng thái

sẵn sàng. Sau đó, quá trình này được đặt vào hàng đợi sẵn sàng. (CPU có thể hay không thể được chuyển từ quá trình đang chạy tới quá trình sẵn sàng mới nhất phụ thuộc vào giải thuật định thời biểu CPU).

Để cài đặt semaphore dưới định nghĩa này, chúng ta định nghĩa một semaphore như một cấu trúc được viết bằng C như sau:

```
typedef struct{

int value;

struct process *L;

} semaphore;
```

Mỗi semaphore có một số integer value và một danh sách các quá trình L. Khi một quá trình phải chờ trên một semaphore, nó được thêm vào danh sách các quá trình L. Một thao tác signal xoá một quá trình ra khỏi danh sách các quá trình đang chờ và đánh thức quá trình đó.

Thao tác semaphore wait bây giờ được định nghĩa như sau:

```
void wait(semaphore S){

S.value--;

If (S.value < 0){

Thêm quá trình này tới danh sách các quá trình S.L;

block();

}

}
```

Thao tác semaphore signal bây giờ có thể được định nghĩa như sau:

```
void signal(semaphore S){
```

```
S.value++;

if(S.value <= 0){

xoá một quá trình ra khỏi hàng đợi S.L;

wakeup(P);

}

}
```

Thao tác block() tạm dừng quá trình gọi thao tác đó. Thao tác wakeup(P) tiếp tục thực thi quá trình bị nghẽn P. Hai thao tác này được cung cấp bởi hệ điều hành như những lời gọi hệ thống cơ bản.

Chú ý rằng, mặc dù dưới sự định nghĩa kinh điển của semaphores với sự chờ đợi bận là giá trị semaphore không bao giờ âm. Cài đặt này có thể có giá trị semaphore âm. Nếu giá trị semaphore âm thì tính chất trọng yếu của nó là số lượng quá trình chờ trên semaphore đó. Sự thật này là kết quả của việc chuyển thứ tự của việc giảm và kiểm tra trong việc cài đặt thao tác wait. Danh sách các quá trình đang chờ có thể được cài đặt dễ dàng bởi một trường liên kết trong mỗi khối điều khiển quá trình (PCB). Mỗi cách thêm và xoá các quá trình từ danh sách, đảm bảo việc chờ đợi có giới hạn sẽ sử dụng hàng đợi FIFO, ở đó semaphore chứa hai con trỏ đầu (head) và đuôi (tail) chỉ tới hàng đợi. Tuy nhiên, danh sách có thể dùng bất cứ chiến lược hàng đợi nào. Sử dụng đúng semaphores không phụ thuộc vào chiến lược hàng đợi cho danh sách semaphore.

Khía cạnh quyết định của semaphores là chúng được thực thi theo tính nguyên tử. Chúng ta phải đảm bảo rằng không có hai quá trình có thể thực thi các thao tác wait và signal trên cùng một semaphore tại cùng một thời điểm. Trường hợp này là vấn đề vùng tương trục và có thể giải quyết bằng một trong hai cách.

Trong môi trường đơn xử lý (nghĩa là chỉ có một CPU tồn tại), đơn giản là chúng ta có thể ngăn chặn các ngắt trong thời gian các thao tác wait và

signal xảy ra. Cơ chế này làm việc trong một môi trường đơn xử lý vì một khi ngắt bị ngăn chặn, các chỉ thị từ các quá trình khác không thể được chen vào. Chỉ quá trình đang chạy hiện tại thực thi cho tới khi các ngắt được cho phép sử dụng trở lại và bộ định thời có thể thu hồi quyền điều khiển.

Trong môi trường đa xử lý, ngăn chặn ngắt không thể thực hiện được. Các chỉ thị từ các quá trình khác nhau (chạy trên các bộ xử lý khác nhau) có thể được chen vào trong cách bất kỳ. Nếu phần cứng không cung cấp bất cứ các chỉ thị đặc biệt nào, chúng ta có thể tận dụng các giải pháp phần cứng phù hợp cho vấn đề vùng tương trục (phần V.-4), ở đó các vùng tương trục chứa cá thủ tục wait và signal.

Vấn đề quan trọng là chúng ta không xoá hoàn toàn chờ đợi bận, với định nghĩa này cho các thao tác wait và signal. Dĩ nhiên là chúng ta xoá chờ đợi bận từ việc đi vào vùng tương trục của chương trình ứng dụng. Ngoài ra, chúng ta hạn chế việc chờ đợi bận chỉ các miền tương trục với thao tác wait và signal và các vùng này là ngắn (nếu được mã hợp lý, chúng nên không quá 10 chỉ thị). Do đó, miền tương trục hầu như không bao giờ bị chiếm và sự chờ đợi bận rất hiếm khi xảy ra và sau đó chỉ cho thời gian ngắn. Một trường hợp hoàn toàn khác xảy ra với những chương trình ứng dụng có miền tương trục dài (vài phút hay thậm chí vài giờ) hay có thể hầu như luôn bị chiếm. Trong trường hợp này, chờ đợi bận là cực kỳ kém hiệu quả.

Sự khoá chết (deadlocks) và đói tài nguyên

Cài đặt semaphore với một hàng đợi có thể dẫn đến trường hợp hai hay nhiều quá trình đang chờ không hạn định một sự kiện mà có thể được gây ra chỉ bởi một trong những quá trình đang chờ. Sự kiện đặt ra là sự thực thi của thao tác signal. Khi một trạng thái như thế xảy ra, những quá trình này được nói là bị khoá chết.

Để hiển thị điều này, chúng ta xét một hệ thống chứa hai quá trình P0 và P1, mỗi truy xuất hai semaphore, S và Q, được đặt giá trị 1.

P0	P1
wait(S);	wait(Q);
wait(Q);	wait(S);
.	.
.	.
signal(S);	signal(Q);
Signal(Q);	signal(S);

Giả sử rằng P0 thực thi wait(S) và sau đó P1 thực thi wait(Q). Khi P0 thực thi wait(Q), nó phải chờ cho đến khi P1 thực thi signal(Q). Tương tự, khi P1 thực thi wait(S), nó phải chờ cho tới khi P0 thực thi signal(S). Vì các thao tác signal này không thể được thực thi nên P0 và P1 bị khoá chết.

Chúng ta nói rằng một tập hợp các quá trình trong trạng thái khoá chết khi mọi quá trình trong tập hợp đang chờ một sự kiện được gây ra chỉ bởi một quá trình khác trong tập hợp. Những sự kiện mà chúng ta quan tâm chủ yếu ở đây là việc chiếm tài nguyên và giải phóng tài nguyên. Tuy nhiên, các loại sự kiện khác cũng có thể dẫn đến việc khoá chết. Chúng ta sẽ xem trong chương VI. Trong chương đó, chúng ta sẽ mô tả các cơ chế khác nhau để giải quyết vấn đề khoá chết.

Một vấn đề khoá chết liên quan tới khoá chết là nghẽn hay đói tài nguyên không hạn định (indefinite blocking or starvation), ở đó các quá trình chờ đợi không hạn định trong semaphore. Nghẽn không hạn định có thể xảy ra nếu chúng ta thêm vào và lấy ra các quá trình từ danh sách được nối kết với một semaphore trong thứ tự vào sau ra trước (LIFO).

Semaphore nhị phân

Xây dựng semaphore được mô tả trong phần trước được gọi là semaphore đếm (counting semaphore) vì giá trị nguyên có thể trãi dài một phạm vi không giới hạn. Một semaphore nhị phân (binary semaphore) là một semaphore với một giá trị nguyên mà trải dài từ 0 và 1. Semaphore nhị phân có thể đơn giản hơn trong cài đặt so với semaphore đếm và phụ thuộc vào kiến trúc phần cứng nằm bên dưới. Chúng sẽ hiển thị cách một semaphore đếm có thể được cài đặt sử dụng semaphore nhị phân dưới đây:

Giả sử S là một semaphore đếm. Để cài đặt nó trong dạng semaphore nhị phân chúng ta cần các cấu trúc dữ liệu như sau:

```
Binary-semaphore S1, S2;

int C;
```

Khởi tạo S1 = 1, S2 = 0 và giá trị nguyên C được đặt tới giá trị khởi tạo của semaphore đếm S.

Thao tác wait trên semaphore đếm S có thể được cài đặt như sau:

```
wait(S);

C--;

If (C<0) {

signal(S1);

wait(S2);

}

signal(S1);
```

Thao tác signal trên semaphore đếm S có thể được cài đặt như sau:

```
wait(S1);
```

C++;

if (C<=0)

signal(S2);

else

signal(S1);

Monitors

Để có thể dễ viết đúng các chương trình đồng bộ hoá hơn, Hoare (1974) và Brinch & Hansen (1975) đề nghị một cơ chế đồng bộ hoá cấp cao hơn được cung cấp bởi ngôn ngữ lập trình là monitor. Một monitor được mô tả bởi một tập hợp của các toán tử được định nghĩa bởi người lập trình. Biểu diễn kiểu của một monitor bao gồm việc khai báo các biến mà giá trị của nó xác định trạng thái của một thể hiện kiểu, cũng như thân của thủ tục hay hàm mà cài đặt các thao tác trên kiểu đó. Cú pháp của monitor được hiển thị trong hình dưới đây:

Monitor <tên monitor>

{

khai báo các biến được chia sẻ

procedure P1 (…){

…

}

procedure P2 (…){

…

}

.

.

.

procedure Pn (…){

…

}

{

mã khởi tạo

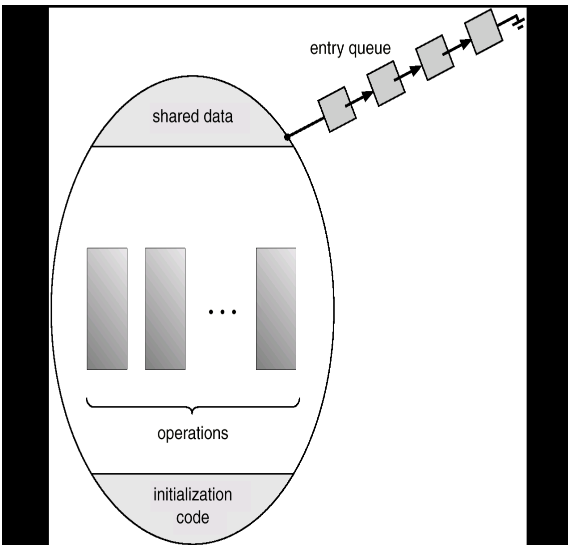
}

}

Hình V-14 Cú pháp của monitor

Biểu diễn kiểu monitor không thể được dùng trực tiếp bởi các quá trình khác nhau. Do đó, một thủ tục được định nghĩa bên trong một monitor chỉ có thể truy xuất những biến được khai báo cục bộ bên trong monitor đó và các tham số chính thức của nó. Tương tự, những biến cục bộ của monitor có thể được truy xuất chỉ bởi những thủ tục cục bộ.

Xây dựng monitor đảm bảo rằng chỉ một quá trình tại một thời điểm có thể được kích hoạt trong monitor. Do đó, người lập trình không cần viết mã ràng buộc đồng bộ hoá như hình V-15 dưới đây:



Hình V-15 Hình ảnh dưới dạng biểu đồ của monitor

Tuy nhiên, xây dựng monitor như được định nghĩa là không đủ mạnh để mô hình hoá các cơ chế đồng bộ. Cho mục đích này, chúng ta cần định nghĩa các cơ chế đồng bộ hoá bổ sung. Những cơ chế này được cung cấp bởi construct condition. Người lập trình có thể định nghĩa một hay nhiều biến của kiểu condition:

condition x, y;

Chỉ những thao tác có thể gọi lên trên các biến điều kiện là wait và signal. Thao tác

x.wait();

có nghĩa là quá trình gọi trên thao tác này được tạm dừng cho đến khi quá trình khác gọi

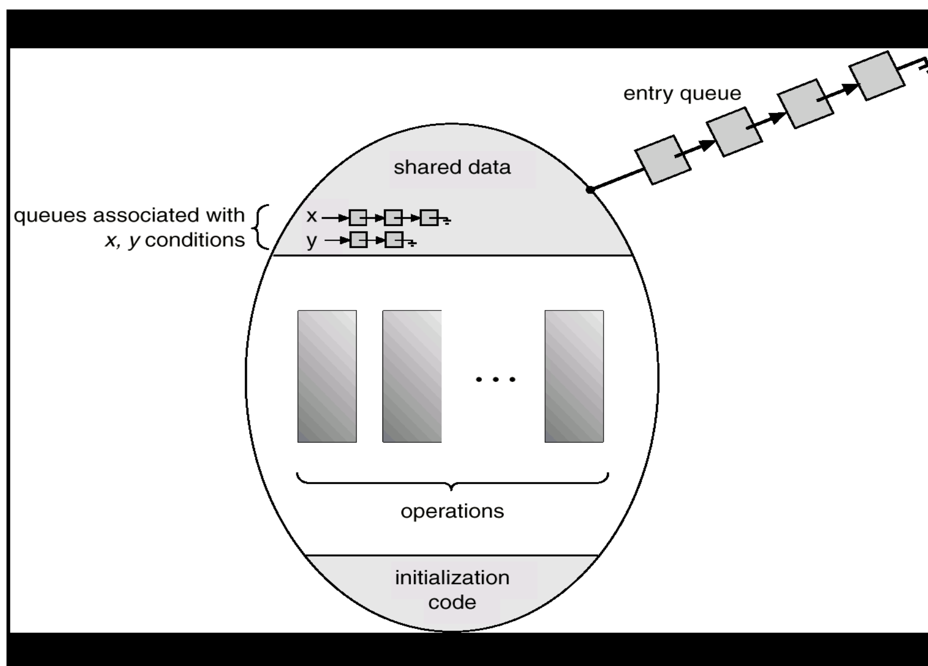
x.signal();

thao tác x.signal() thực thi tiếp một cách chính xác một quá trình tạm dừng. Nếu không có quá trình tạm dừng thì thao tác signal không bị ảnh hưởng gì cả; nghĩa là trạng thái x như thể thao tác chưa bao giờ được

thực thi (như hình V.-16). Ngược lại, với thao tác signal được gán cùng với semaphores luôn ảnh hưởng tới trạng thái của semaphore.

Bây giờ giả sử rằng, khi thao tác x.signal() được gọi bởi một quá trình P thì có một quá trình Q gán với biến điều kiện x bị tạm dừng. Rõ ràng, nếu quá trình Q được phép thực thi tiếp thì quá trình P phải dừng. Nếu không thì cả hai quá trình P và Q hoạt động cùng một lúc trong monitor. Tuy nhiên, về khái niệm hai quá trình có thể tiếp tục việc thực thi của chúng. Hai khả năng có thể xảy ra:

1. P chờ cho đến khi Q rời khỏi monitor hoặc chờ điều kiện khác.
2. Q chờ cho đến khi P rời monitor hoặc chờ điều kiện khác.



Hình V-16 Monitor với các biến điều kiện

Có các luận cứ hợp lý trong việc chấp nhận khả năng 1 hay 2. Vì P đã thực thi trong monitor rồi, nên chọn khả năng 2 có vẻ hợp lý hơn. Tuy nhiên, nếu chúng ta cho phép quá trình P tiếp tục, biến điều kiện "luận lý" mà Q đang chờ có thể không còn quản lý thời gian Q được tiếp tục. Chọn khả năng 1 được tán thành bởi Hoare vì tham số đầu tiên của nó

chuyển trực tiếp tới các qui tắc chứng minh đơn giản hơn. Thoả hiệp giữa hai khả năng này được chấp nhận trong ngôn ngữ đồng hành C. Khi quá trình P thực thi thao tác signal thì quá trình Q lập tức được tiếp tục. Mô hình này không mạnh hơn mô hình của Hoare vì một quá trình không thể báo hiệu nhiều lần trong một lời gọi thủ tục đơn.

Bây giờ chúng ta xem xét cài đặt cơ chế monitor dùng semaphores. Đối với mỗi monitor, một biến semaphore mutex (được khởi tạo 1) được cung cấp. Một quá trình phải thực thi wait(mutex) trước khi đi vào monitor và phải thực thi signal(mutex) sau khi rời monitor.

Vì quá trình đang báo hiệu phải chờ cho đến khi quá trình được bắt đầu lại rời hay chờ, một biến semaphore bổ sung next được giới thiệu, được khởi tạo 0 trên quá trình báo hiệu có thể tự tạm dừng. Một biến số nguyên next_count cũng sẽ được cung cấp để đếm số lượng quá trình bị tạm dừng trên next. Do đó, mỗi thủ tục bên ngoài F sẽ được thay thế bởi

wait(mutex);

. . .

thân của F

if (next_count > 0)

signal(next);

else

signal(mutex);

Loại trừ hỗ tương trong monitor được đảm bảo.

Bây giờ chúng ta mô tả các biến điều kiện được cài đặt như thế nào. Đối với mỗi biến điều kiện x, chúng ta giới thiệu một biến semaphore x_sem và biến số nguyên x_count, cả hai được khởi tạo tới 0. Thao tác x.wait có thể được cài đặt như sau:

```
x_count++;

if ( next_count > 0)

signal(next);

else

signal(mutex);

wait(x_sem);

x_count--;
```

Thao tác x.signal() có thể được cài đặt như sau:

```
if ( x_count > 0){

next_count++;

signal(x_sem);

wait(next);

next_count--;

}
```

Cài đặt này có thể áp dụng để định nghĩa của monitor được cho bởi cả hai Hoare và Brinch Hansen. Tuy nhiên, trong một số trường hợp tính tổng quát của việc cài đặt là không cần thiết và yêu cầu có một cải tiến hiệu quả hơn.

Bây giờ chúng ta sẽ trở lại chủ đề thứ tự bắt đầu lại của quá trình trong monitor. Nếu nhiều quá trình bị trì hoãn trên biến điều kiện x và thao tác x.signal được thực thi bởi một vài quá trình thì thứ tự các quá trình bị trì hoãn được thực thi trở lại như thế nào? Một giải pháp đơn giản là dùng thứ tự FCFS vì thế quá trình chờ lâu nhất sẽ được thực thi tiếp trước. Tuy nhiên, trong nhiều trường hợp, cơ chế định thời biểu như thế là

không đủ. Cho mục đích này cấu trúc conditional-wait có thể được dùng; nó có dạng

x.wait(c);

ở đây c là một biểu thức số nguyên được định giá khi thao tác wait được thực thi. Giá trị c, được gọi là số ưu tiên, được lưu với tên quá trình được tạm dừng. Khi x.signal được thực thi, quá trình với số ưu tiên nhỏ nhất được thực thi tiếp.

Để hiển thị cơ chế mới này, chúng ta xem xét monitor được hiển thị như hình dưới đây, điều khiển việc cấp phát của một tài nguyên đơn giữa các quá trình cạnh tranh. Mỗi quá trình khi yêu cầu cấp phát tài nguyên của nó, xác định thời gian tối đa nó hoạch định để sử dụng tài nguyên. Monitor cấp phát tài nguyên tới quá trình có yêu cầu thời gian cấp phát ngắn nhất.

	Monitor ResourceAllocation{boolean busy;conditionx;void acquire(int time){if (busy) x.wait(time);busy = true;}void release(){busy = false;x.signal();}void init(){busy = false;}}

Hình V-17 Một monitor cấp phát tới một tài nguyên

Một quá trình cần truy xuất tài nguyên phải chú ý thứ tự sau:

R.acquire(t);

…

truy xuất tài nguyên

...

R.release();

ở đây R là thể hiện của kiểu ResourceAllocation.

Tuy nhiên, khái niệm monitor không đảm bảo rằng các thứ tự truy xuất trước sẽ được chú ý. Đặc biệt,

- Một quá trình có thể truy xuất tài nguyên mà không đạt được quyền truy xuất trước đó.
- Một quá trình sẽ không bao giờ giải phóng tài nguyên một khi nó được gán truy xuất tới tài nguyên đó.
- Một quá trình có thể cố gắng giải phóng tài nguyên mà nó không bao giờ yêu cầu.
- Một quá trình có thể yêu cầu cùng tài nguyên hai lần (không giải phóng tài nguyên đó trong lần đầu)

Việc sử dụng monitor cũng gặp cùng những khó khăn như xây dựng miền tương trục. Trong phần trước, chúng ta lo lắng về việc sử dụng đúng semaphore. Bây giờ, chúng ta lo lắng về việc sử dụng đúng các thao tác được định nghĩa của người lập trình cấp cao mà các trình biên dịch không còn hỗ trợ chúng ta.

Một giải pháp có thể đối với vấn đề trên là chứa các thao tác truy xuất tài nguyên trong monitor ResourceAllocation. Tuy nhiên, giải pháp này sẽ dẫn đến việc định thời được thực hiện dựa theo giải thuật định thời monitor được xây dựng sẵn hơn là được viết bởi người lập trình.

Để đảm bảo rằng các quá trình chú ý đến thứ tự hợp lý, chúng ta phải xem xét kỹ tất cả chương trình thực hiện việc dùng monitor ResourceAllocation và những tài nguyên được quản lý của chúng. Có hai điều kiện mà chúng ta phải kiểm tra để thiết lập tính đúng đắn của hệ thống. Đầu tiên, các quá trình người dùng phải luôn luôn thực hiện các lời gọi của chúng trên monitor trong thứ tự đúng. Thứ hai, chúng ta phải đảm bảo rằng một quá trình không hợp tác không đơn giản bỏ qua cổng (gateway) loại trừ hỗ tương được cung cấp bởi monitor và cố gắng truy xuất trực tiếp tài nguyên được chia sẻ mà không sử dụng giao thức truy xuất. Chỉ nếu hai điều kiện này có thể được đảm bảo có thể chúng ta

đảm bảo rằng không có lỗi ràng buộc thời gian nào xảy ra và giải thuật định thời sẽ không bị thất bại.

Mặc dù việc xem xét này có thể cho hệ thống nhỏ, tĩnh nhưng nó không phù hợp cho một hệ thống lớn hay động. Vấn đề kiểm soát truy xuất có thể được giải quyết chỉ bởi một cơ chế bổ sung khác.

Các bài toán đồng bộ hoá nguyên thuỷ

Trong phần này, chúng ta trình bày một số bài toán đồng bộ hoá như những thí dụ về sự phân cấp lớn các vấn đề điều khiển đồng hành. Các vấn đề này được dùng cho việc kiểm tra mọi cơ chế đồng bộ hoá được đề nghị gần đây. Semaphore được dùng cho việc đồng bộ hoá trong các giải pháp dưới đây.

Bài toán người sản xuất-người tiêu thụ

Bài toán người sản xuất-người tiêu thụ (Producer-Consumer) thường được dùng để hiển thị sức mạnh của các hàm cơ sở đồng bộ hoá. Hai quá trình cùng chia sẻ một vùng đệm có kích thước giới hạn n. Biến semaphore mutex cung cấp sự loại trừ hỗ tương để truy xuất vùng đệm và được khởi tạo với giá trị 1. Các biến semaphore empty và full đếm số khe trống và đầy tương ứng. Biến semaphore empty được khởi tạo tới giá trị n; biến semaphore full được khởi tạo tới giá trị 0.

Mã cho người quá trình sản xuất được hiển thị trong hình V.-18:

do{…sản xuất sản phẩm trong nextp…wait(empty);wait(mutex);…thêm nextp tới vùng đệm…signal(mutex);signal(full);} while (1);

Hình V-18 Cấu trúc của quá trình người sản xuất

Mã cho quá trình người tiêu thụ được hiển thị trong hình dưới đây:

do{wait(full);wait(mutex);…lấy một sản phẩm từ vùng đệm tới nextc… signal(mutex);signal(empty);} while (1);

Hình V-19 Cấu trúc của quá trình người tiêu thụ

Bài toán bộ đọc-bộ ghi

Bộ đọc-bộ ghi (Readers-Writers) là một đối tượng dữ liệu (như một tập tin hay mẫu tin) được chia sẻ giữa nhiều quá trình đồng hành. Một số trong các quá trình có thể chỉ cần đọc nội dung của đối tượng được chia sẻ, ngược lại một vài quá trình khác cần cập nhật (nghĩa là đọc và ghi) trên đối tượng được chia sẻ. Chúng ta phân biệt sự khác nhau giữa hai loại quá trình này bằng cách gọi các quá trình chỉ đọc là bộ đọc và các quá trình cần cập nhật là bộ ghi. Chú ý, nếu hai bộ đọc truy xuất đối tượng được chia sẻ cùng một lúc sẽ không có ảnh hưởng gì. Tuy nhiên, nếu một bộ ghi và vài quá trình khác (có thể là bộ đọc hay bộ ghi) truy xuất cùng một lúc có thể dẫn đến sự hỗn độn.

Để đảm bảo những khó khăn này không phát sinh, chúng ta yêu cầu các bộ ghi có truy xuất loại trừ lẫn nhau tới đối tượng chia sẻ. Việc đồng bộ hoá này được gọi là bài toán bộ đọc-bộ ghi. Bài toán bộ đọc-bộ ghi có một số biến dạng liên quan đến độ ưu tiên. Dạng đơn giản nhất là bài toán bộ đọc trước-bộ ghi (first reader-writer). Trong dạng này yêu cầu không có bộ đọc nào phải chờ ngoại trừ có một bộ ghi đã được cấp quyền sử dụng đối tượng chia sẻ. Nói cách khác, không có bộ đọc nào phải chờ các bộ đọc khác để hoàn thành đơn giản vì một bộ ghi đang chờ. Bài toán bộ đọc sau-bộ ghi (second readers-writers) yêu cầu một khi bộ ghi đang sẵn sàng, bộ ghi đó thực hiện việc ghi của nó sớm nhất có thể. Nói một cách khác, nếu bộ ghi đang chờ truy xuất đối tượng, không có bộ đọc nào có thể bắt đầu việc đọc.

Giải pháp cho bài toán này có thể dẫn đến việc đói tài nguyên. Trong trường hợp đầu, các bộ ghi có thể bị đói; trong trường hợp thứ hai các bộ đọc có thể bị đói. Trong giải pháp cho bài toán bộ đọc trước-bộ ghi, các quá trình bộ đọc chia sẻ các cấu trúc dữ liệu sau:

semaphore mutex, wrt;

int readcount;

Biến semaphore mutex và wrt được khởi tạo 1; biến readcount được khởi tạo 0. Biến semaphore wrt dùng chung cho cả hai quá trình bộ đọc và bộ ghi. Biến semaphore mutex được dùng để đảm bảo loại trừ hỗ tương khi biến readcount được cập nhật. Biến readcount ghi vết có bao nhiêu quá trình hiện hành đang đọc đối tượng. Biến semaphore wrt thực hiện chức năng như một biến semaphore loại trừ hỗ tương cho các bộ đọc. Nó cũng được dùng bởi bộ đọc đầu tiên hay bộ đọc cuối cùng mà nó đi vào hay thoát khỏi miền tương trục. Nó cũng không được dùng bởi các bộ đọc mà nó đi vào hay thoát trong khi các bộ đọc khác đang ở trong miền tương trục.

Mã cho quá trình bộ viết được hiển thị như hình V.-20:

wait(wrt);…Thao tác viết được thực hiệnsignal(wrt);

Hình V-20 Cấu trúc của quá trình viết

Mã của quá trình đọc được hiển thị như hình V.-21:

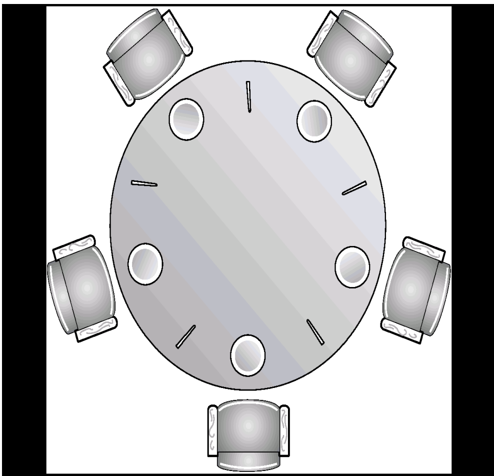
wait(mutex);readcount++;if (readcount == 1)wait(wrt);signal(mutex);… Thao tác đọc được thực hiệnwait(mutex);readcount--;if (readcount == 0)signal(wrt);signal(mutex);

Hình V-21 Cấu trúc của bộ đọc

Chú ý rằng, nếu bộ viết đang ở trong miền tương trục và n bộ đọc đang chờ thì một bộ đọc được xếp hàng trên wrt, và n-1 được xếp hàng trên mutex. Cũng cần chú ý thêm, khi một bộ viết thực thi signal(wrt) thì chúng ta có thể thực thi tiếp việc thực thi của các quá trình đọc đang chờ hay một quá trình viết đang chờ. Việc chọn lựa này có thể được thực hiện bởi bộ định thời biểu.

Bài toán các triết gia ăn tối

Có năm nhà triết gia, vừa suy nghĩ vừa ăn tối. Các triết gia ngồi trên cùng một bàn tròn xung quanh có năm chiếc ghế, mỗi chiếc ghế được ngồi bởi một triết gia. Chính giữa bàn là một bát cơm và năm chiếc đũa được hiển thị như hình VI-22:



Hình V-22 Tình huống các triết gia ăn tối

Khi một triết gia suy nghĩ, ông ta không giao tiếp với các triết gia khác. Thỉnh thoảng, một triết gia cảm thấy đói và cố gắng chọn hai chiếc đũa gần nhất (hai chiếc đũa nằm giữa ông ta với hai láng giềng trái và phải). Một triết gia có thể lấy chỉ một chiếc đũa tại một thời điểm. Chú ý, ông ta không thể lấy chiếc đũa mà nó đang được dùng bởi người láng giềng. Khi một triết gia đói và có hai chiếc đũa cùng một lúc, ông ta ăn mà không đặt đũa xuống. Khi triết gia ăn xong, ông ta đặt đũa xuống và bắt đầu suy nghĩ tiếp.

Bài toán các triết gia ăn tối được xem như một bài toán đồng bộ hoá kinh điển. Nó trình bày yêu cầu cấp phát nhiều tài nguyên giữa các quá trình trong cách tránh việc khoá chết và đói tài nguyên.

Một giải pháp đơn giản là thể hiện mỗi chiếc đũa bởi một biến semaphore. Một triết gia cố gắng chiếm lấy một chiếc đũa bằng cách

thực thi thao tác wait trên biến semaphore đó; triết gia đặt hai chiếc đũa xuống bằng cách thực thi thao tác signal trên các biến semaphore tương ứng. Do đó, dữ liệu được chia sẻ là:

semaphore chopstick[5];

ở đây tất cả các phần tử của chopstick được khởi tạo 1. Cấu trúc của philosopher i được hiển thị như hình dưới đây:

do{wait(chopstick[i]);wait(chopstick[(i + 1) % 5]);…ăn… signal(chopstick[i]);signal(chopstick[(i + 1) % 5]);…suy nghĩ…} while (1);

Hình V-23 Cấu trúc của triết gia thứ i

Mặc dù giải pháp này đảm bảo rằng không có hai láng giềng nào đang ăn cùng một lúc nhưng nó có khả năng gây ra khoá chết. Giả sử rằng năm triết gia bị đói cùng một lúc và mỗi triết gia chiếm lấy chiếc đũa bên trái của ông ta. Bây giờ tất cả các phần tử chopstick sẽ là 0. Khi mỗi triết gia cố gắng dành lấy chiếc đũa bên phải, triết gia sẽ bị chờ mãi mãi.

Nhiều giải pháp khả thi đối với vấn đề khoá chết được liệt kê tiếp theo. Giải pháp cho vấn đề các triết gia ăn tối mà nó đảm bảo không bị khoá chết.

- Cho phép nhiều nhất bốn triết gia đang ngồi cùng một lúc trên bàn
- Cho phép một triết gia lấy chiếc đũa của ông ta chỉ nếu cả hai chiếc đũa là sẵn dùng (để làm điều này ông ta phải lấy chúng trong miền tương trục).
- Dùng một giải pháp bất đối xứng; nghĩa là một triết gia lẽ chọn đũa bên trái đầu tiên của ông ta và sau đó đũa bên phải, trái lại một triết gia chẳn chọn chiếc đũa bên phải và sau đó chiếc đũa bên phải của ông ta.

Tóm lại, bất cứ một giải pháp nào thoả mãn đối với bài toán các triết gia ăn tối phải đảm bảo dựa trên khả năng một trong những triết gia sẽ đói

chết. Giải pháp giải quyết việc khoá chết không cần thiết xoá đi khả năng đói tài nguyên.

Tóm tắt

Một tập hợp các quá trình tuần tự cộng tác chia sẻ dữ liệu, loại trừ hỗ tương phải được cung cấp. Một giải pháp đảm bảo rằng vùng tương trục của mã đang sử dụng chỉ bởi một quá trình hay một luồng tại một thời điểm. Các giải thuật khác tồn tại để giải quyết vấn đề miền tương trục, với giả thuyết rằng chỉ khoá bên trong việc lưu trữ là sẵn dùng.

Sự bất lợi chủ yếu của các giải pháp được mã hoá bởi người dùng là tất cả chúng đều yêu cầu sự chờ đợi bận. Semaphore khắc phục sự bất lợi này. Semaphores có thể được dùng để giải quyết các vấn đề đồng bộ khác nhau và có thể được cài đặt hiệu quả, đặc biệt nếu phần cứng hỗ trợ các thao tác nguyên tử.

Các bài toán đồng bộ khác (chẳng hạn như bài toán người sản xuất-người tiêu dùng, bài toán bộ đọc, bộ ghi và bài toán các triết gia ăn tối) là cực kỳ quan trọng vì chúng là thí dụ của phân lớp lớn các vấn đề điều khiển đồng hành. Vấn đề này được dùng để kiểm tra gần như mọi cơ chế đồng bộ được đề nghị gần đây.

Hệ điều hành phải cung cấp phương tiện để đảm bảo chống lại lỗi thời gian. Nhiều cấu trúc dữ liệu được đề nghị để giải quyết các vấn đề này. Các vùng tương trục có thể được dùng để cài đặt loại trừ hỗ tương và các vấn đề đồng bộ an toàn và hiệu quả. Monitors cung cấp cơ chế đồng bộ cho việc chia sẻ các loại dữ liệu trừu tượng. Một biến điều kiện cung cấp một phương thức cho một thủ tục monitor khoá việc thực thi của nó cho đến khi nó được báo hiệu tiếp tục.

Deadlock
1 Mục đích Sau khi học xong chương này, người học nắm được những kiến thức sau: - Hiểu mô hình hệ thống về deadlock - Hiểu các đặc điểm của deadlock - Hiểu các phương pháp quản lý deadlock - Hiểu cách ngăn chặn deadlock - Hiểu cách tránh deadlock - Hiểu cách phát hiện deadlock - Hiểu cách phục hồi từ deadlock

Giới thiệu

Trong môi trường đa chương, nhiều quá trình có thể cạnh tranh một số giới hạn tài nguyên. Một quá trình yêu cầu tài nguyên, nếu tài nguyên không sẵn dùng tại thời điểm đó, quá trình đi vào trạng thái chờ. Quá trình chờ có thể không bao giờ chuyển trạng thái trở lại vì tài nguyên chúng yêu cầu bị giữ bởi những quá trình đang chờ khác. Trường hợp này được gọi là deadlock (khoá chết).

Trong chương này chúng ta sẽ mô tả các phương pháp mà hệ điều hành có thể dùng để ngăn chặn hay giải quyết deadlock. Hầu hết các hệ điều hành không cung cấp phương tiện ngăn chặn deadlock nhưng những đặc điểm này sẽ được thêm vào sau đó. Vấn đề deadlock chỉ có thể trở thành vấn đề phổ biến, xu hướng hiện hành gồm số lượng lớn quá trình, chương trình đa luồng, nhiều tài nguyên trong hệ thống và đặc biệt các tập tin có đời sống dài và những máy phục vụ cơ sở dữ liệu hơn là các hệ thống bó.

Mô hình hệ thống

Một hệ thống chứa số tài nguyên hữu hạn được phân bổ giữa nhiều quá trình cạnh tranh. Các tài nguyên này được phân chia thành nhiều loại, mỗi loại chứa một số thể hiện xác định. Không gian bộ nhớ, các chu kỳ CPU và các thiết bị nhập/xuất (như máy in, đĩa từ) là những thí dụ về loại tài nguyên. Nếu hệ thống có hai CPUs, thì loại tài nguyên CPU có hai thể hiện. Tương tự, loại tài nguyên máy in có thể có năm thể hiện.

Nếu một quá trình yêu cầu một thể hiện của loại tài nguyên thì việc cấp phát bất cứ thể hiện nào của loại tài nguyên này sẽ thoả mãn yêu cầu. Nếu nó không có thì các thể hiện là không xác định và các lớp loại tài

nguyên sẽ không được định nghĩa hợp lý. Thí dụ, một hệ thống có thể có hai máy in. Hai loại máy in này có thể được định nghĩa trong cùng lớp loại tài nguyên nếu không có quá trình nào quan tâm máy nào in ra dữ liệu. Tuy nhiên, nếu một máy in ở tầng 9 và máy in khác ở tầng trệt thì người dùng ở tầng 9 không thể xem hai máy in là tương tự nhau và lớp tài nguyên riêng rẻ cần được định nghĩa cho mỗi máy in.

Một quá trình phải yêu cầu một tài nguyên trước khi sử dụng nó, và phải giải phóng sau khi sử dụng nó. Một quá trình có thể yêu cầu nhiều tài nguyên như nó được yêu cầu để thực hiện tác vụ được gán của nó. Chú ý, số tài nguyên được yêu cầu không vượt quá số lượng tổng cộng tài nguyên sẵn có trong hệ thống. Nói cách khác, một quá trình không thể yêu cầu ba máy in nếu hệ thống chỉ có hai.

Dưới chế độ điều hành thông thường, một quá trình có thể sử dụng một tài nguyên chỉ trong thứ tự sau:

Yêu cầu: nếu yêu cầu không thể được gán tức thì (thí dụ, tài nguyên đang được dùng bởi quá trình khác) thì quá trình đang yêu cầu phải chờ cho tới khi nó có thể nhận được tài nguyên.

Sử dụng: quá trình có thể điều hành tài nguyên (thí dụ, nếu tài nguyên là máy in, quá trình có thể in máy in)

Giải phóng: quá trình giải phóng tài nguyên.

Yêu cầu và giải phóng tài nguyên là các lời gọi hệ thống. Thí dụ như yêu cầu và giải phóng thiết bị, mở và đóng tập tin, cấp phát và giải phóng bộ nhớ. Yêu cầu và giải phóng các tài nguyên khác có thể đạt được thông qua thao tác chờ wait và báo hiệu signal. Do đó, cho mỗi trường hợp sử dụng, hệ điều hành kiểm tra để đảm bảo rằng quá trình sử dụng yêu cầu và được cấp phát tài nguyên. Một bảng hệ thống ghi nhận mỗi quá trình giải phóng hay được cấp phát tài nguyên. Nếu một quá trình yêu cầu tài nguyên mà tài nguyên đó hiện được cấp phát cho một quá trình khác, nó có thể được thêm vào hàng đợi để chờ tài nguyên này.

Một tập hợp quá trình trong trạng thái deadlock khi mỗi quá trình trong tập hợp này chờ sự kiện mà có thể được tạo ra chỉ bởi quá trình khác trong tập hợp. Những sự kiện mà chúng ta quan tâm chủ yếu ở đây là nhận và giải phóng tài nguyên. Các tài nguyên có thể là tài nguyên vật lý (thí dụ, máy in, đĩa từ, không gian bộ nhớ và chu kỳ CPU) hay tài nguyên luận lý (thí dụ, tập tin, semaphores, monitors). Tuy nhiên, các loại khác của sự kiện có thể dẫn đến deadlock.

Để minh hoạ trạng thái deadlock, chúng ta xét hệ thống với ba ổ đĩa từ. Giả sử mỗi quá trình giữ các một ổ đĩa từ này. Bây giờ, nếu mỗi quá trình yêu cầu một ổ đĩa từ khác thì ba quá trình sẽ ở trong trạng thái deadlock. Mỗi quá trình đang chờ một sự kiện "ổ đĩa từ được giải phóng" mà có thể được gây ra chỉ bởi một trong những quá trình đang chờ. Thí dụ này minh hoạ deadlock liên quan đến cùng loại tài nguyên.

Deadlock cũng liên quan nhiều loại tài nguyên khác nhau. Thí dụ, xét một hệ thống với một máy in và một ổ đĩa từ. Giả sử, quá trình Pi đang giữ ổ đĩa từ và quá trình Pj đang giữ máy in. Nếu Pi yêu cầu máy in và Pj yêu cầu ổ đĩa từ thì deadlock xảy ra.

Một người lập trình đang phát triển những ứng dụng đa luồng phải quan tâm đặc biệt tới vấn đề này: Các chương trình đa luồng là ứng cử viên cho vấn đề deadlock vì nhiều luồng có thể cạnh tranh trên tài nguyên được chia sẻ.

Đặc điểm deadlock

Trong một deadlock, các quá trình không bao giờ hoàn thành việc thực thi và các tài nguyên hệ thống bị buộc chặt, ngăn chặn các quá trình khác bắt đầu. Trước khi chúng ta thảo luận các phương pháp khác nhau giải quyết vấn đề deadlock, chúng ta sẽ mô tả các đặc điểm mà deadlock mô tả.

Những điều kiện cần thiết gây ra deadlock

Trường hợp deadlock có thể phát sinh nếu bốn điều kiện sau xảy ra cùng một lúc trong hệ thống:

1. Loại trừ hỗ tương: ít nhất một tài nguyên phải được giữ trong chế độ không chia sẻ; nghĩa là, chỉ một quá trình tại cùng một thời điểm có thể sử dụng tài nguyên. Nếu một quá trình khác yêu cầu tài nguyên đó, quá trình yêu cầu phải tạm dừng cho đến khi tài nguyên được giải phóng.
2. Giữ và chờ cấp thêm tài nguyên: quá trình phải đang giữ ít nhất một tài nguyên và đang chờ để nhận tài nguyên thêm mà hiện đang được giữ bởi quá trình khác.
3. Không đòi lại tài nguyên từ quá trình đang giữ chúng: Các tài nguyên không thể bị đòi lại; nghĩa là, tài nguyên có thể được giải phóng chỉ tự ý bởi quá trình đang giữ nó, sau khi quá trình đó hoàn thành tác vụ.
4. Tồn tại chu trình trong đồ thị cấp phát tài nguyên: một tập hợp các quá trình {P0, P1,…,Pn} đang chờ mà trong đó P0 đang chờ một tài nguyên được giữ bởi P1, P1 đang chờ tài nguyên đang giữ bởi P2,…,Pn-1 đang chờ tài nguyên đang được giữ bởi quá trình P0.

Chúng ta nhấn mạnh rằng tất cả bốn điều kiện phải cùng phát sinh để deadlock xảy ra. Điều kiện chờ đợi ch trình đưa đến điều kiện giữ-và-chờ vì thế bốn điều kiện không hoàn toàn độc lập.

Đồ thị cấp phát tài nguyên

Deadlock có thể mô tả chính xác hơn bằng cách hiển thị đồ thị có hướng gọi là đồ thị cấp phát tài nguyên hệ thống. Đồ thị này chứa một tập các đỉnh V và tập hợp các cạnh E. Một tập các đỉnh V được chia làm hai loại nút P = {P1, P2,…,Pn} là tập hợp các quá trình hoạt động trong hệ thống, và R = {R1, R2, ..., Rm} là tập hợp chứa tất cả các loại tài nguyên trong hệ thống.

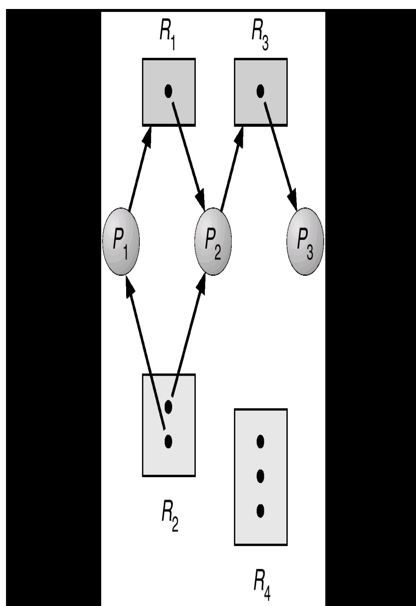
Một cạnh có hướng từ quá trình Pi tới loại tài nguyên Rj được ký hiệu Pi Rj; nó biểu thị rằng quá trình Pi đã yêu cầu loại tài nguyên Rj và hiện đang chờ loại tài nguyên đó. Một cạnh có hướng từ loại tài nguyên Rj tới quá

trình Pi được hiển thị bởi Rj Pi; nó hiển thị rằng thể hiện của loại tài nguyên Rj đã được cấp phát tới quá trình Pi. Một cạnh có hướng Pi Rj được gọi là cạnh yêu cầu; một cạnh có hướng Rj Pi được gọi là cạnh gán.

Bằng hình tượng, chúng ta hiển thị mỗi quá trình Pi là một hình tròn, và mỗi loại tài nguyên Rj là hình chữ nhật. Vì loại tài nguyên Rj có thể có nhiều hơn một thể hiện, chúng ta hiển thị mỗi thể hiện là một chấm nằm trong hình vuông. Chú ý rằng một cạnh yêu cầu trỏ tới chỉ một hình vuông Rj, trái lại một cạnh gán cũng phải gán tới một trong các dấu chấm trong hình vuông.

Khi quá trình Pi yêu cầu một thể hiện của loại tài nguyên Rj, một cạnh yêu cầu được chèn vào đồ thị cấp phát tài nguyên. Khi yêu cầu này có thể được đáp ứng, cạnh yêu cầu lập tức được truyền tới cạnh gán. Khi quá trình không còn cần truy xuất tới tài nguyên, nó giải phóng tài nguyên, và khi đó dẫn đến cạnh gán bị xoá.

Đồ thị cấp phát tài nguyên được hiển thị trong hình VI-1 dưới đây mô tả trường hợp sau:



Hình VI-1 Đồ thị cấp phát tài nguyên

- Các tập P, R, và E:
    - P = {P1, P2, P3}
    - R = {R1, R2, R3, R4}
    - E = {P1 R1, P2 R3, R1 P2, R2 P2, R3 P3}
- Các thể hiện tài nguyên
    - Một thể hiện của tài nguyên loại R1
    - Hai thể hiện của tài nguyên loại R2
    - Một thể hiện của tài nguyên loại R3
    - Một thể hiện của tài nguyên loại R4
- Trạng thái quá trình
    - Quá trình P1 đang giữ một thể hiện của loại tài nguyên R2 và đang chờ một thể hiện của loại tài nguyên R1.
    - Quá trình P2 đang giữ một thể hiện của loại tài nguyên R1 và R2 và đang chờ một thể hiện của loại tài nguyên R3.
    - Quá trình P3 đang giữ một thể hiện của R3

Đồ thị cấp phát tài nguyên hiển thị rằng, nếu đồ thị không chứa chu trình, thì không có quá trình nào trong hệ thống bị deadlock. Nếu đồ thị có chứa chu trình, thì deadlock có thể tồn tại.

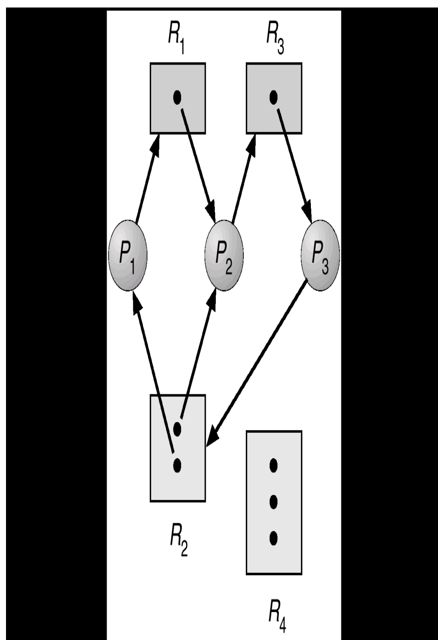
Nếu mỗi loại tài nguyên có chính xác một thể hiện, thì một chu trình ngụ ý rằng một deadlock xảy ra. Nếu một chu trình bao gồm chỉ một tập hợp các loại tài nguyên, mỗi loại tài nguyên chỉ có một thể hiện thì deadlock xảy ra. Mỗi quá trình chứa trong chu trình bị deadlock. Trong trường hợp này, một chu trình trong đồ thị là điều kiện cần và đủ để tồn tại deadlock.

Nếu mỗi loại tài nguyên có nhiều thể hiện thì chu trình không ngụ ý deadlock xảy. Trong trường hợp này, một chu trình trong đồ thị là điều kiện cần nhưng chưa đủ để tồn tại deadlock.

Để hiển thị khái niệm này, chúng ta xem lại đồ thị ở hình VII-1 ở trên. Giả sử quá trình P3 yêu cầu một thể hiện của loại tài nguyên R2. Vì không có thể hiện tài nguyên hiện có, một cạnh yêu cầu P3   R2 được thêm vào đồ thị (hình VI-2). Tại thời điểm này, hai chu trình nhỏ tồn tại trong hệ thống:

P1   R1   P2   R3   P3   R2   P1

P2   R3   P3   R2   P2

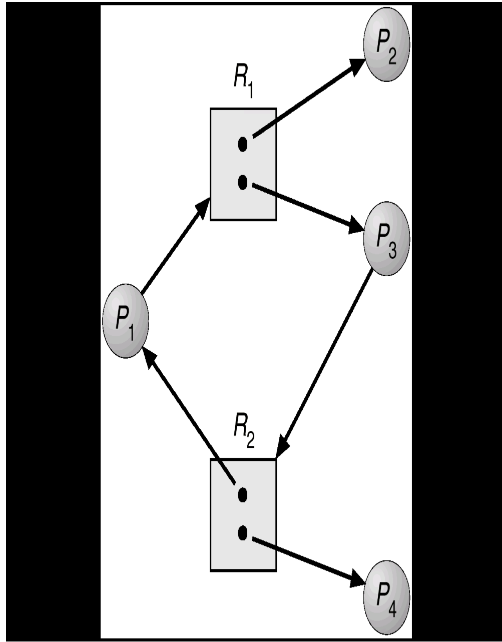


Hình VI-2 Đồ thị cấp phát tài nguyên với deadlock

Quá trình P1, P2, và P3 bị deadlock. Quá trình P3 đang chờ tài nguyên R3, hiện được giữ bởi quá trình P2. Hay nói cách khác, quá trình P3 đang chờ quá trình P1 hay P2 giải phóng tài nguyên R2. Ngoài ra, quá trình P1 đang chờ quá trình P2 giải phóng tài nguyên R1.

Bây giờ xem xét đồ thị cấp phát tài nguyên trong hình VI-3 dưới đây. Trong thí dụ này, chúng ta cũng có một chu kỳ

P1   R1   P3   R2   P1



Hình VI-3 Đồ thị cấp phát tài nguyên có chu trình nhưng không bị deadlock

Tuy nhiên, không có deadlock. Chú ý rằng quá trình P4 có thể giải phóng thể hiện của loại tài nguyên R2. Tài nguyên đó có thể được cấp phát tới P3 sau đó, chu trình sẽ không còn.

Tóm lại, nếu đồ thị cấp phát tài nguyên không có chu trình thì hệ thống không có trạng thái deadlock. Ngoài ra, nếu có chu trình thì có thể có hoặc không trạng thái deadlock. Nhận xét này là quan trọng khi chúng ta giải quyết vấn đề deadlock.

Các phương pháp xử lý deadlock

Phần lớn, chúng ta có thể giải quyết vấn đề deadlock theo một trong ba cách:

- Chúng ta có thể sử dụng một giao thức để ngăn chặn hay tránh deadlocks, đảm bảo rằng hệ thống sẽ không bao giờ đi vào trạng thái deadlock
- Chúng ta có thể cho phép hệ thống đi vào trạng thái deadlock, phát hiện nó và phục hồi.

- Chúng ta có thể bỏ qua hoàn toàn vấn đề này và giả vờ deadlock không bao giờ xảy ra trong hệ thống. Giải pháp này được dùng trong nhiều hệ điều hành, kể cả UNIX.

Chúng ta sẽ tìm hiểu vắn tắt mỗi phương pháp. Sau đó, chúng ta sẽ trình bày các giải thuật một cách chi tiết trong các phần sau đây.

Để đảm bảo deadlock không bao giờ xảy ra, hệ thống có thể dùng kế hoạch ngăn chặn hay tránh deadlock. Ngăn chặn deadlock là một tập hợp các phương pháp để đảm bảo rằng ít nhất một điều kiện cần (trong phần VI.4.1) không thể xảy ra. Các phương pháp này ngăn chặn deadlocks bằng cách ràng buộc yêu cầu về tài nguyên được thực hiện như thế nào. Chúng ta thảo luận phương pháp này trong phần sau.

Ngược lại, tránh deadlock yêu cầu hệ điều hành cung cấp những thông tin bổ sung tập trung vào loại tài nguyên nào một quá trình sẽ yêu cầu và sử dụng trong thời gian sống của nó. Với những kiến thức bổ sung này, chúng ta có thể quyết định đối với mỗi yêu cầu quá trình nên chờ hay không. Để quyết định yêu cầu hiện tại có thể được thoả mãn hay phải bị trì hoãn, hệ thống phải xem xét tài nguyên hiện có, tài nguyên hiện cấp phát cho mỗi quá trình, và các yêu cầu và giải phóng tương lai của mỗi quá trình.

Nếu một hệ thống không dùng giải thuật ngăn chặn hay tránh deadlock thì trường hợp deadlock có thể xảy ra. Trong môi trường này, hệ thống có thể cung cấp một giải thuật để xem xét trạng thái của hệ thống để xác định deadlock có xảy ra hay không và giải thuật phục hồi từ deadlock.

Nếu hệ thống không đảm bảo rằng deadlock sẽ không bao giờ xảy ra và cũng không cung cấp một cơ chế để phát hiện và phục hồi deadlock thì có thể dẫn đến trường hợp hệ thống ở trong trạng thái deadlock. Trong trường hợp này, deadlock không được phát hiện sẽ làm giảm năng lực hệ thống vì tài nguyên đang được giữ bởi những quá trình mà chúng không thể thực thi, đi vào trạng thái deadlock. Cuối cùng, hệ thống sẽ dừng các chức năng và cần được khởi động lại bằng thủ công.

Mặc dù phương pháp này dường như không là tiếp cận khả thi đối với vấn đề deadlock nhưng nó được dùng trong một số hệ điều hành. Trong

nhiều hệ thống, deadlock xảy ra không thường xuyên; do đó phương pháp này là rẻ hơn chi phí cho phương pháp ngăn chặn deadlock, tránh deadlock, hay phát hiện và phục hồi deadlock mà chúng phải được sử dụng liên tục. Trong một số trường hợp, hệ thống ở trong trạng thái cô đặc nhưng không ở trạng thái deadlock. Như thí dụ, xem xét một quá trình thời thực chạy tại độ ưu tiên cao nhất (hay bất cứ quá trình đang chạy trên bộ định thời biểu không trưng dụng) và không bao giờ trả về điều khiển đối với hệ điều hành. Do đó, hệ thống phải có phương pháp phục hồi bằng thủ công cho các điều kiện không deadlock và có thể đơn giản sử dụng các kỹ thuật đó cho việc phục hồi deadlock.

Ngăn chặn deadlock

Để deadlock xảy ra, một trong bốn điều kiện cần phải xảy ra. Bằng cách đảm bảo ít nhất một trong bốn điều kiện này không thể xảy ra, chúng ta có thể ngăn chặn việc xảy ra của deadlock. Chúng ta tìm hiểu tỷ mỹ tiếp cận này bằng cách xem xét mỗi điều kiện cần riêng rẻ nhau.

Loại trừ hỗ tương

Điều kiện loại trừ hỗ tương phải giữ cho tài nguyên không chia sẻ. Thí dụ, một máy in không thể được chia sẻ cùng lúc bởi nhiều quá trình. Ngược lại, các tài nguyên có thể chia sẻ không đòi hỏi truy xuất loại trừ hỗ tương và do đó không thể liên quan đến deadlock. Những tập tin chỉ đọc là một thí dụ tốt cho tài nguyên có thể chia sẻ. Nếu nhiều quá trình cố gắng mở một tập tin chỉ đọc tại cùng một thời điểm thì chúng có thể được gán truy xuất cùng lúc tập tin. Một quá trình không bao giờ yêu cầu chờ tài nguyên có thể chia sẻ. Tuy nhiên, thường chúng ta không thể ngăn chặn deadlock bằng cách từ chối điều kiện loại trừ hỗ tương: một số tài nguyên về thực chất không thể chia sẻ.

Giữ và chờ cấp thêm tài nguyên

Để đảm bảo điều kiện giữ-và-chờ cấp thêm tài nguyên không bao giờ xảy ra trong hệ thống, chúng ta phải đảm bảo rằng bất cứ khi nào một quá trình yêu cầu tài nguyên, nó không giữ bất cứ tài nguyên nào khác. Một giao thức có thể được dùng là đòi hỏi mỗi quá trình yêu cầu và được cấp

phát tất cả tài nguyên trước khi nó bắt đầu thực thi. Chúng ta có thể cài đặt sự cung cấp này bằng cách yêu cầu các lời gọi hệ thống yêu cầu tài nguyên cho một quá trình trước tất cả các lời gọi hệ thống khác.

Một giao thức khác cho phép một quá trình yêu cầu tài nguyên chỉ khi quá trình này không có tài nguyên nào. Một quá trình có thể yêu cầu một số tài nguyên và dùng chúng. Tuy nhiên, trước khi nó có thể yêu cầu bất kỳ tài nguyên bổ sung nào, nó phải giải phóng tất cả tài nguyên mà nó hiện đang được cấp phát.

Để hiển thị sự khác nhau giữa hai giao thức, chúng ta xét một quá trình chép dữ liệu từ băng từ tới tập tin đĩa, sắp xếp tập tin đĩa và sau đó in kết quả ra máy in. Nếu tất cả tài nguyên phải được yêu cầu cùng một lúc thì khởi đầu quá trình phải yêu cầu băng từ, tập tin đĩa và máy in. Nó sẽ giữ máy in trong toàn thời gian thực thi của nó mặc dù nó cần máy in chỉ ở giai đoạn cuối.

Phương pháp thứ hai cho phép quá trình yêu cầu ban đầu chỉ băng từ và tập tin đĩa. Nó chép dữ liệu từ băng từ tới đĩa, rồi giải phóng cả hai băng từ và đĩa. Sau đó, quá trình phải yêu cầu lại tập tin đĩa và máy in. Sau đó, chép tập tin đĩa tới máy in, nó giải phóng hai tài nguyên này và kết thúc.

Hai giao thức này có hai nhược điểm chủ yếu. Thứ nhất, việc sử dụng tài nguyên có thể chậm vì nhiều tài nguyên có thể được cấp nhưng không được sử dụng trong thời gian dài. Trong thí dụ được cho, chúng ta có thể giải phóng băng từ và tập tin đĩa, sau đó yêu cầu lại tập tin đĩa và máy in chỉ nếu chúng ta đảm bảo rằng dữ liệu của chúng ta sẽ vẫn còn trên tập tin đĩa. Nếu chúng ta không thể đảm bảo rằng dữ liệu vẫn còn tập tin đĩa thì chúng ta phải yêu cầu tất cả tài nguyên tại thời điểm bắt đầu cho cả hai giao thức. Thứ hai, đói tài nguyên là có thể. Một quá trình cần nhiều tài nguyên phổ biến có thể phải đợi vô hạn định vì một tài nguyên mà nó cần luôn được cấp phát cho quá trình khác.

Không đòi lại tài nguyên từ quá trình đang giữ chúng

Điều kiện cần thứ ba là không đòi lại những tài nguyên đã được cấp phát rồi. Để đảm bảo điều kiện này không xảy ra, chúng ta có thể dùng giao

thức sau. Nếu một quá trình đang giữ một số tài nguyên và yêu cầu tài nguyên khác mà không được cấp phát tức thì tới nó (nghĩa là, quá trình phải chờ) thì tất cả tài nguyên hiện đang giữ được đòi lại. Nói cách khác, những tài nguyên này được giải phóng hoàn toàn. Những tài nguyên bị đòi lại được thêm tới danh sách các tài nguyên mà quá trình đang chờ. Quá trình sẽ được khởi động lại chỉ khi nó có thể nhận lại tài nguyên cũ của nó cũng như các tài nguyên mới mà nó đang yêu cầu.

Có một sự chọn lựa khác, nếu một quá trình yêu cầu một số tài nguyên, đầu tiên chúng ta kiểm tra chúng có sẵn không. Nếu tài nguyên có sẵn, chúng ta cấp phát chúng. Nếu tài nguyên không có sẵn, chúng ta kiểm tra chúng có được cấp phát tới một số quá trình khác đang chờ tài nguyên bổ sung. Nếu đúng như thế, chúng ta lấy lại tài nguyên mong muốn đó từ quá trình đang đợi và cấp chúng cho quá trình đang yêu cầu. Nếu tài nguyên không sẵn có hay được giữ bởi một quá trình đang đợi, quá trình đang yêu cầu phải chờ. Trong khi nó đang chờ, một số tài nguyên của nó có thể được đòi lại chỉ nếu quá trình khác yêu cầu chúng. Một quá trình có thể được khởi động lại chỉ khi nó được cấp các tài nguyên mới mà nó đang yêu cầu và phục hồi bất cứ tài nguyên nào đã bị lấy lại trong khi nó đang chờ.

Giao thức này thường được áp dụng tới tài nguyên mà trạng thái của nó có thể được lưu lại dễ dàng và phục hồi lại sau đó, như các thanh ghi CPU và không gian bộ nhớ. Nó thường không thể được áp dụng cho các tài nguyên như máy in và băng từ.

Tồn tại chu trình trong đồ thị cấp phát tài nguyên

Điều kiện thứ tư và cũng là điều kiện cuối cùng cho deadlock là điều kiện tồn tại chu trình trong đồ thị cấp phát tài nguyên. Một cách để đảm bảo rằng điều kiện này không bao giờ xảy ra là áp đặt toàn bộ thứ tự của tất cả loại tài nguyên và đòi hỏi mỗi quá trình trong thứ tự tăng của số lượng.

Gọi R = {R1, R2, …, Rm} là tập hợp loại tài nguyên. Chúng ta gán mỗi loại tài nguyên một số nguyên duy nhất, cho phép chúng ta so sánh hai tài nguyên và xác định tài nguyên này có đứng trước tài nguyên khác hay không trong thứ tự của chúng ta. Thông thường, chúng ta định nghĩa hàm ánh xạ một-một F: R   N, ở đây N là tập hợp các số tự nhiên. Thí dụ, nếu tập hợp

các loại tài nguyên R gồm các ổ băng từ, ổ đĩa và máy in thì hàm F có thể được định nghĩa như sau:

F(ổ băng từ) = 1,

F(đĩa từ) = 5,

F(máy in) = 12.

Bây giờ chúng ta xem giao thức sau để ngăn chặn deadlock: mỗi quá trình có thể yêu cầu tài nguyên chỉ trong thứ tự tăng của số lượng. Nghĩa là, một quá trình ban đầu có thể yêu cầu bất cứ số lượng thể hiện của một loại tài nguyên Ri. Sau đó, một quá trình có thể yêu cầu các thể hiện của loại tài nguyên Rj nếu và chỉ nếu F(Rj) > F(Ri). Nếu một số thể hiện của cùng loại tài nguyên được yêu cầu, thì một yêu cầu cho tất cả thể hiện phải được cấp phát. Thí dụ, sử dụng hàm được định nghĩa trước đó, một quá trình muốn dùng ổ băng từ và máy in tại cùng một lúc trước tiên phải yêu cầu ổ băng từ và sau đó yêu cầu máy in.

Nói một cách khác, chúng ta yêu cầu rằng, bất cứ khi nào một quá trình yêu cầu một thể hiện của loại tài nguyên Rj, nó giải phóng bất cứ tài nguyên Ri sao cho F(Ri)   F(Rj).

Nếu có hai giao thức được dùng thì điều kiện tồn tại chu trình không thể xảy ra. Chúng ta có thể giải thích điều này bằng cách cho rằng tồn tại chu trình trong đồ thị cấp phát tài nguyên tồn tại. Gọi tập hợp các quá trình chứa tồn tại chu trình trong đồ thị cấp phát tài nguyên là {P0, P1, … , Pn}, ở đây Pi đang chờ một tài nguyên Ri, mà Ri được giữ bởi quá trình Pi+1. Vì sau đó quá trình Pi+1 đang giữ tài nguyên Ri trong khi yêu cầu tài nguyên Ri+1, nên chúng ta có F(Ri) < F(Ri+1) cho tất cả i. Nhưng điều kiện này có nghĩa là F(R0) < F(R1) < …< F(Rn) < F(R0). Bằng qui tắc bắt cầu F(R0) < F(R0), điều này là không thể. Do đó, không thể có chờ chu trình.

Chú ý rằng hàm F nên được định nghĩa dựa theo thứ tự tự nhiên của việc sử dụng tài nguyên trong hệ thống. Thí dụ, vì ổ băng từ thường được yêu cầu trước máy in nên có thể hợp lý để định nghĩa F(ổ băng từ) < F(máy in).

Tránh deadlock

Các giải thuật ngăn chặn deadlock, được thảo luận ở VII-6, ngăn chặn deadlock bằng cách hạn chế cách các yêu cầu có thể được thực hiện. Các ngăn chặn đảm bảo rằng ít nhất một trong những điều kiện cần cho deadlock không thể xảy ra. Do đó, deadlock không thể xảy ra. Tuy nhiên, các tác dụng phụ có thể ngăn chặn deadlock bởi phương pháp này là việc sử dụng thiết bị chậm và thông lượng hệ thống bị giảm.

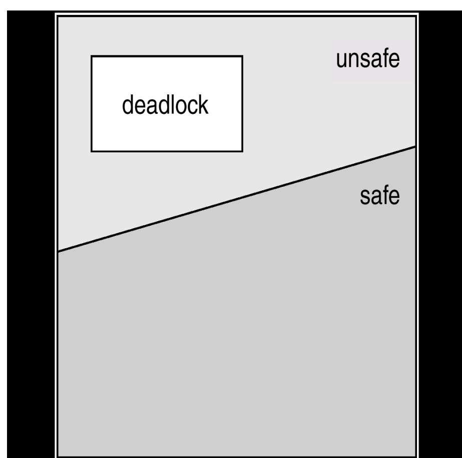
Một phương pháp khác để tránh deadlock là yêu cầu thông tin bổ sung về cách tài nguyên được yêu cầu. Thí dụ, trong một hệ thống với một ổ băng từ và một máy in, chúng ta có thể bảo rằng quá trình P sẽ yêu cầu ổ băng từ trước và sau đó máy in trước khi giải phóng cả hai tài nguyên. Trái lại, quá trình Q sẽ yêu cầu máy in trước và sau đó ổ băng từ. Với kiến thức về thứ tự hoàn thành của yêu cầu và giải phóng cho mỗi quá trình, chúng ta có thể quyết định cho mỗi yêu cầu của quá trình sẽ chờ hay không. Mỗi yêu cầu đòi hỏi hệ thống xem tài nguyên hiện có, tài nguyên hiện được cấp tới mỗi quá trình, và các yêu cầu và giải phóng tương lai của mỗi quá trình, để yêu cầu của quá trình hiện tại có thể được thoả mãn hay phải chờ để tránh khả năng xảy ra deadlock.

Các giải thuật khác nhau có sự khác nhau về lượng và loại thông tin được yêu cầu. Mô hình đơn giản và hữu ích nhất yêu cầu mỗi quá trình khai báo số lớn nhất tài nguyên của mỗi loại mà nó cần. Thông tin trước về số lượng tối đa tài nguyên của mỗi loại được yêu cầu cho mỗi quá trình, có thể xây dựng một giải thuật đảm bảo hệ thống sẽ không bao giờ đi vào trạng thái deadlock. Đây là giải thuật định nghĩa tiếp cận tránh deadlock. Giải thuật tránh deadlock tự xem xét trạng thái cấp phát tài nguyên để đảm bảo điều kiện tồn tại chu trình trong đồ thị cấp phát tài nguyên có thể không bao giờ xảy ra. Trạng thái cấp phát tài nguyên được định nghĩa bởi số tài nguyên sẵn dùng và tài nguyên được cấp phát và số yêu cầu tối đa của các quá trình.

Trạng thái an toàn

Một trạng thái là an toàn nếu hệ thống có thể cấp phát các tài nguyên tới mỗi quá trình trong một vài thứ tự và vẫn tránh deadlock. Hay nói cách khác, một hệ thống ở trong trạng thái an toàn chỉ nếu ở đó tồn tại một thứ tự an toàn. Thứ tự của các quá trình <P1, P2, …, Pn> là một thứ tự an toàn cho trạng thái cấp phát hiện hành nếu đối với mỗi thứ tự Pi, các tài nguyên mà Pi yêu cầu vẫn có thể được thoả mãn bởi tài nguyên hiện có cộng với các tài nguyên được giữ bởi tất cả Pj, với j<i. Trong trường hợp này, nếu những tài nguyên mà quá trình Pi yêu cầu không sẵn dùng tức thì thì Pi có thể chờ cho đến khi tất cả Pj hoàn thành. Khi chúng hoàn thành, Pi có thể đạt được tất cả những tài nguyên nó cần, hoàn thành các tác vụ được gán, trả về những tài nguyên được cấp phát cho nó và kết thúc. Khi Pi kết thúc, Pi+1 có thể đạt được các tài nguyên nó cần,... Nếu không có thứ tự như thế tồn tại thì trạng thái hệ thống là không an toàn.

Một trạng thái an toàn không là trạng thái deadlock. Do đó, trạng thái deadlock là trạng thái không an toàn. Tuy nhiên, không phải tất cả trạng thái không an toàn là deadlock (hình VI-4). Một trạng thái không an toàn có thể dẫn đến deadlock. Với điều kiện trạng thái là an toàn, hệ điều hành có thể tránh trạng thái không an toàn (và deadlock). Trong một trạng thái không an toàn, hệ điều hành có thể ngăn chặn các quá trình từ những tài nguyên đang yêu cầu mà deadlock xảy ra: hành vi của các quá trình này điều khiển các trạng thái không an toàn.



Hình VI-4 Không gian trạng thái an toàn, không an toàn, deadlock

Để minh hoạ, chúng ta xét một hệ thống với 12 ổ băng từ và 3 quá trình: P-0, P1, P2. Quá trình P0 yêu cầu 10 ổ băng từ, quá trình P1 có thể cần 4 và quá trình P2 có thể cần tới 9 ổ băng từ. Giả sử rằng tại thời điểm t0, quá trình P0 giữ 5 ổ băng từ, quá trình P1 giữ 2 và quá trình P2 giữ 2 ổ băng từ. (Do đó, có 3 ổ băng từ còn rảnh).

	Nhu cầu tối đa	Nhu cầu hiện tại
P0	10	5
P1	4	2
P2	9	2

Tại thời điểm t0, hệ thống ở trạng thái an toàn. Thứ tự <P1,P0, P2> thoả điều kiện an toàn vì quá trình P1 có thể được cấp phát tức thì tất cả các ổ đĩa từ và sau đó trả lại chúng (sau đó hệ thống có 5 ổ băng từ sẵn dùng), sau đó quá trình P0 có thể nhận tất cả ổ băng từ và trả lại chúng (sau đó hệ thống sẽ có 10 ổ băng từ sẵn dùng), và cuối cùng quá trình P2 có thể nhận tất cả ổ băng từ của nó và trả lại chúng (sau đó hệ thống sẽ có tất cả 12 ổ băng từ sẵn dùng).

Một hệ thống có thể đi từ trạng thái an toàn tới một trạng thái không an toàn. Giả sử rằng tại thời điểm t1, quá trình P2 yêu cầu và được cấp 1 ổ băng từ nữa. Hệ thống không còn trong trạng thái an toàn. Tại điểm này, chỉ quá trình P1 có thể được cấp tất cả ổ băng từ của nó. Khi nó trả lại chúng, chỉ quá trình P1 có thể được cấp phát tất cả ổ băng từ. Khi nó trả lại chúng, hệ thống chỉ còn 4 ổ băng từ sẳn có. Vì quá trình P0 được cấp phát 5 ổ băng từ, nhưng có tối đa 10, quá trình P0 phải chờ. Tương tự, quá trình P2 có thể yêu cầu thêm 6 ổ băng từ và phải chờ dẫn đến deadlock.

Lỗi của chúng ta là gán yêu cầu từ quá trình P2 cho 1 ổ băng từ nữa. Nếu chúng ta làm cho P2 phải chờ cho đến khi các quá trình khác kết thúc và giải phóng tài nguyên của nó thì chúng ta có thể tránh deadlock.

Với khái niệm trạng thái an toàn được cho, chúng ta có thể định nghĩa các giải thuật tránh deadlock. Ý tưởng đơn giản là đảm bảo hệ thống sẽ luôn còn trong trạng thái an toàn. Khởi đầu, hệ thống ở trong trạng thái an toàn. Bất cứ khi nào một quá trình yêu cầu một tài nguyên hiện có, hệ thống phải quyết định tài nguyên có thể được cấp phát tức thì hoặc quá trình phải chờ. Yêu cầu được gán chỉ nếu việc cấp phát để hệ thống trong trạng thái an toàn.

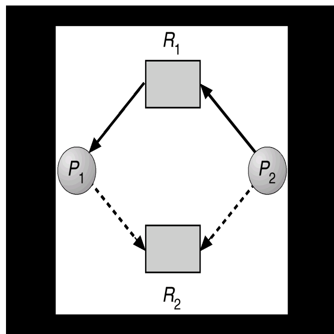
Trong mô hình này, nếu quá trình yêu cầu tài nguyên đang có, nó có thể vẫn phải chờ. Do đó, việc sử dụng tài nguyên có thể chậm hơn mà không có giải thuật tránh deadlock.

Giải thuật đồ thị cấp phát tài nguyên

Nếu chúng ta có một hệ thống cấp phát tài nguyên với một thể hiện của mỗi loại, một biến dạng của đồ thị cấp phát tài nguyên được định nghĩa trong phần VI.4.2 có thể được dùng để tránh deadlock.

Ngoài các cạnh yêu cầu và gán, chúng ta giới thiệu một loại cạnh mới được gọi là cạnh thỉnh cầu (claim edge). Một cạnh thỉnh cầu Pi   Rj hiển thị quá trình Pi có thể yêu cầu tài nguyên Rj vào một thời điểm trong tương lai. Cạnh này tương tự cạnh yêu cầu về phương hướng nhưng được hiện diện bởi dấu đứt khoảng. Khi quá trình Pi yêu cầu tài nguyên Rj, cạnh thỉnh cầu Pi   Rj chuyển tới cạnh yêu cầu. Tương tự, khi một tài nguyên Rj được giải phóng bởi Pi, cạnh gán Rj   Pi được chuyển trở lại thành cạnh thỉnh cầu Pi   Rj. Chúng ta chú ý rằng các tài nguyên phải được yêu cầu trước trong hệ thống. Nghĩa là, trước khi Pi bắt đầu thực thi, tất cả các cạnh thỉnh cầu của nó phải xuất hiện trong đồ thị cấp phát tài nguyên. Chúng ta có thể giảm nhẹ điều kiện này bằng cách cho phép một cạnh Pi Rj để được thêm tới đồ thị chỉ nếu tất cả các cạnh gắn liền với quá trình Pi là các cạnh thỉnh cầu.

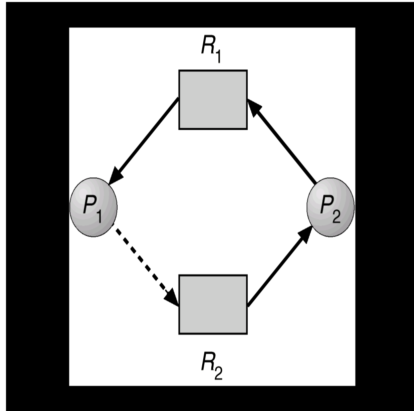
Giả sử rằng Pi yêu cầu tài nguyên Rj. Yêu cầu có thể được gán chỉ nếu chuyển cạnh yêu cầu Pi Rj tới cạnh gán Rj Pi không dẫn đến việc hình thành chu trình trong đồ thị cấp phát tài nguyên. Chú ý rằng chúng ta kiểm tra tính an toàn bằng cách dùng giải thuật phát hiện chu trình. Một giải thuật để phát hiện một chu trình trong đồ thị này yêu cầu một thứ tự của n2 thao tác, ở đây n là số quá trình trong hệ thống.



Hình VI-5 Đồ thị cấp phát tài nguyên để tránh deadlock

Nếu không có chu trình tồn tại, thì việc cấp phát tài nguyên sẽ để lại hệ thống trong trạng thái an toàn. Nếu chu trình được tìm thấy thì việc cấp phát sẽ đặt hệ thống trong trạng thái không an toàn. Do đó, quá trình Pi sẽ phải chờ yêu cầu của nó được thoả.

Để minh hoạ giải thuật này, chúng ta xét đồ thị cấp phát tài nguyên của hình VI-5. Giả sử rằng P2 yêu cầu R2. Mặc dù R2 hiện rảnh nhưng chúng ta không thể cấp phát nó tới P2 vì hoạt động này sẽ tạo ra chu trình trong đồ thị (Hình VI-6). Một chu trình hiển thị rằng hệ thống ở trong trạng thái không an toàn. Nếu P1 yêu cầu R2 và P2 yêu cầu R1 thì deadlock sẽ xảy ra.



Hình VI-6 Trạng thái không an toàn trong đồ thị cấp phát tài nguyên

Giải thuật của Banker

Giải thuật đồ thị cấp phát tài nguyên không thể áp dụng tới hệ thống cấp phát tài nguyên với nhiều thể hiện của mỗi loại tài nguyên. Giải thuật tránh deadlock mà chúng ta mô tả tiếp theo có thể áp dụng tới một hệ thống nhưng ít hiệu quả hơn cơ chế đồ thị cấp phát tài nguyên. Giải thuật này thường được gọi là giải thuật của Banker. Tên được chọn vì giải thuật này có thể được dùng trong hệ thống ngân hàng để đảm bảo ngân hàng không bao giờ cấp phát tiền mặt đang có của nó khi nó không thể thoả mãn các yêu cầu của tất cả khách hàng.

Khi một quá trình mới đưa vào hệ thống, nó phải khai báo số tối đa các thể hiện của mỗi loại tài nguyên mà nó cần. Số này có thể không vượt quá tổng số tài nguyên trong hệ thống. Khi một người dùng yêu cầu tập hợp các tài nguyên, hệ thống phải xác định việc cấp phát của các tài nguyên này sẽ để lại hệ thống ở trạng thái an toàn hay không. Nếu trạng thái hệ thống sẽ là an toàn, tài nguyên sẽ được cấp; ngược lại quá trình phải chờ cho tới khi một vài quá trình giải phóng đủ tài nguyên.

Nhiều cấu trúc dữ liệu phải được duy trì để cài đặt giải thuật Banker. Những cấu trúc dữ liệu này mã hoá trạng thái của hệ thống cấp phát tài nguyên. Gọi n là số quá trình trong hệ thống và m là số loại tài nguyên trong hệ thống. Chúng ta cần các cấu trúc dữ liệu sau:

- Available: một vector có chiều dài m hiển thị số lượng tài nguyên sẵn dùng của mỗi loại. Nếu Available[j]= k, có k thể hiện của loại tài nguyên Rj sẵn dùng.
- Max: một ma trận n x m định nghĩa số lượng tối đa yêu cầu của mỗi quá trình. Nếu Max[i , j] = k, thì quá trình Pi có thể yêu cầu nhiều nhất k thể hiện của loại tài nguyên Rj.
- Allocation: một ma trận n x m định nghĩa số lượng tài nguyên của mỗi loại hiện được cấp tới mỗi quá trình. Nếu Allocation[i, j] = k, thì quá trình Pi hiện được cấp k thể hiện của loại tài nguyên Rj.
- Need: một ma trận n x m hiển thị yêu cầu tài nguyên còn lại của mỗi quá trình. Nếu Need[i, j] = k, thì quá trình Pi có thể cần thêm k thể hiện của loại tài nguyên Rj để hoàn thành tác vụ của nó. Chú ý rằng, Need[i, j] = Max[i, j] – Allocation [i, j].

Cấu trúc dữ liệu này biến đổi theo thời gian về kích thước và giá trị

Để đơn giản việc trình bày của giải thuật Banker, chúng ta thiết lập vài ký hiệu. Gọi X và Y là các vector có chiều dài n. Chúng ta nói rằng X Y nếu và chỉ nếu X[i] Y[i] cho tất cả i = 1, 2, …, n. Thí dụ, nếu X = (1, 7, 3, 2) và Y = (0, 3, 2, 1) thì Y X, Y < X nếu Y X và Y X.

Chúng ta có thể xem xét mỗi dòng trong ma trận Allocation và Need như là những vectors và tham chiếu tới chúng như Allocationi và Needi tương ứng. Vector Allocationi xác định tài nguyên hiện được cấp phát tới quá trình Pi; vector Needi xác định các tài nguyên bổ sung mà quá trình Pi có thể vẫn yêu cầu để hoàn thành tác vụ của nó.

Giải thuật an toàn

Giải thuật để xác định hệ thống ở trạng thái an toàn hay không có thể được mô tả như sau:

1. Gọi Work và Finish là các vector có chiều dài m và n tương ứng. Khởi tạo Work:=Available và Finish[i]:=false cho i = 1, 2, …,n.
2. Tìm i thỏa:
    - Finish[i] = false
    - Need i Work.

Nếu không có i nào thỏa, di chuyển tới bước 4

1. Work:=Work + Allocation i

Finish[i] := true

Di chuyển về bước 2.

1. Nếu Finish[i] = true cho tất cả i, thì hệ thống đang ở trạng thái an toàn.

Giải thuật này có thể yêu cầu độ phức tạp mxn2 thao tác để quyết định trạng thái là an toàn hay không.

Giải thuật yêu cầu tài nguyên

Cho Requesti là vector yêu cầu cho quá trình Pi. Nếu Requesti[j] = k, thì quá trình Pi muốn k thể hiện của loại tài nguyên Rj. Khi một yêu cầu tài nguyên được thực hiện bởi quá trình Pi, thì các hoạt động sau được thực hiện:

1. Nếu Requesti   Needi, di chuyển tới bước 2. Ngược lại, phát sinh một điều kiện lỗi vì quá trình vượt quá yêu cầu tối đa của nó.
2. Nếu Requesti   Available, di chuyển tới bước 3. Ngược lại, Pi phải chờ vì tài nguyên không sẵn có.
3. Giả sử hệ thống cấp phát các tài nguyên được yêu cầu tới quá trình Pi bằng cách thay đổi trạng thái sau:

Available := Available – Requesti;

Allocationi := Allocationi + Requesti;

Needi := Needi – Requesti;

Nếu kết quả trạng thái cấp phát tài nguyên là an toàn, thì giao dịch được hoàn thành và quá trình Pi được cấp phát tài nguyên của nó. Tuy nhiên, nếu trạng thái mới là không an toàn, thì Pi phải chờ Requesti và trạng thái cấp phát tài nguyên cũ được phục hồi.

Thí dụ minh họa

Xét một hệ thống với 5 quá trình từ P0 tới P4, và 3 loại tài nguyên A, B, C. Loại tài nguyên A có 10 thể hiện, loại tài nguyên B có 5 thể hiện và loại tài nguyên C có 7 thể hiện. Giả sử rằng tại thời điểm T0 trạng thái hiện tại của hệ thống như sau:

	Allocation			Max			Available		
	A	B	C	A	B	C	A	B	C
P0	0	1	0	7	5	3	3	3	2
P1	2	0	0	3	2	2			
P2	3	0	2	9	0	2			
P3	2	1	1	2	2	2			
P4	0	0	2	4	3	3			

Nội dung ma trận Need được định nghĩa là Max-Allocation và là

	Need		
	A	B	C
P0	7	4	3
P1	1	2	2
P2	6	0	2
P3	0	1	1
P4	4	3	1

Chúng ta khẳng định rằng hệ thống hiện ở trong trạng thái an toàn. Thật vậy, thứ tự <P1, P3, P4, P2, P0> thỏa tiêu chuẩn an toàn. Giả sử bây giờ P1 yêu cầu thêm một thể hiện loại A và hai thể hiện loại C, vì thế Request1 = (1, 0, 2). Để quyết định yêu cầu này có thể được cấp tức thì hay không, trước tiên chúng ta phải kiểm tra Request1 Available (nghĩa là, (1, 0, 2)) (3, 3, 2)) là đúng hay không. Sau đó, chúng ta giả sử yêu cầu này đạt được và chúng ta đi đến trạng thái mới sau:

	Allocation			Max			Available		
	A	B	C	A	B	C	A	B	C
P0	0	1	0	7	4	3	2	3	0
P1	3	0	2	0	2	0			
P2	3	0	2	6	0	0			
P3	2	1	1	0	1	1			
P4	0	0	2	4	3	1			

Chúng ta phải xác định trạng thái mới này là an toàn hay không. Để thực hiện điều này, chúng ta thực thi giải thuật an toàn của chúng ta và tìm thứ tự <P1, P3, P4, P0, P2> thỏa yêu cầu an toàn. Do đó, chúng ta có thể cấp lập tức yêu cầu của quá trình P1.

Tuy nhiên, chúng ta cũng thấy rằng, khi hệ thống ở trong trạng thái này, một yêu cầu (3, 3, 0) bởi P4 không thể được gán vì các tài nguyên là không sẵn dùng. Một yêu cầu cho (0, 2, 0) bởi P0 không thể được cấp mặc dù tài nguyên là sẵn dùng vì trạng thái kết quả là không an toàn.

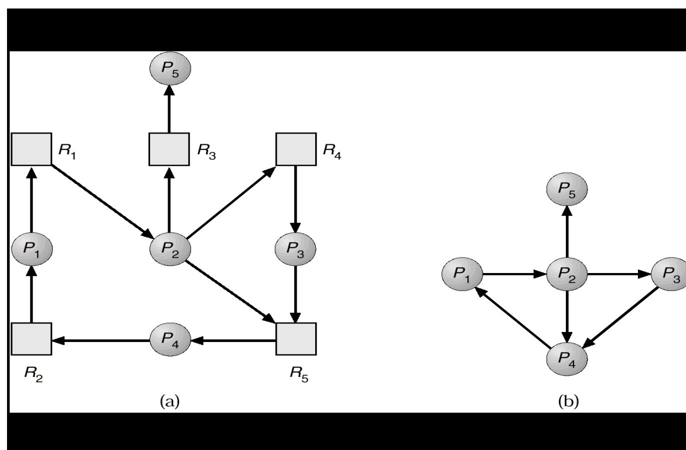
Phát hiện Deadlock

Nếu một hệ thống không thực hiện giải thuật ngăn chặn deadlock hay tránh deadlock thì trường hợp deadlock có thể xảy ra. Trong môi trường này, hệ thống phải cung cấp:

- Giải thuật xem xét trạng thái của hệ thống để quyết định deadlock có xảy ra hay không.
- Giải thuật phục hồi từ deadlockTrong thảo luận dưới đây, chúng ta thảo luận chi tiết về hai yêu cầu khi chúng liên quan đến những hệ thống với chỉ một thể hiện của mỗi loại tài nguyên cũng như đối với

hệ thống có nhiều thể hiện cho mỗi loại tài nguyên. Tuy nhiên, tại thời điểm này chúng ta chú ý lược đồ phát hiện và phục hồi yêu cầu chi phí bao gồm không chỉ chi phí tại thời điểm thực thi cho việc duy trì thông tin cần thiết và thực thi giải thuật phát hiện mà còn các lãng phí có thể phát sinh trong việc phát hiện từ deadlock.

Một thể hiện của mỗi loại tài nguyên



Nếu tất cả tài nguyên chỉ có một thể hiện thì chúng ta có thể định nghĩa giải thuật phát hiện deadlock dùng một biến dạng của đồ thị cấp phát tài nguyên, được gọi là đồ thị chờ (wait-for). Chúng ta đạt được đồ thị này từ đồ thị cấp phát tài nguyên bằng cách gỡ bỏ các nút của loại tài nguyên và xóa các cạnh tương ứng.

Hình VI-7 a) Đồ thị cấp phát tài nguyên. b) Đồ thị chờ tương ứng

Chính xác hơn, một cạnh từ Pi tới Pj trong đồ thị chờ hiển thị rằng quá trình Pi đang chờ một quá trình Pj để giải phóng tài nguyên mà Pi cần. Cạnh Pi   Pj tồn tại trong đồ thị chờ nếu và chỉ nếu đồ thị cấp phát tài nguyên tương ứng chứa hai cạnh Pi   Rq và Rq   Pj đối với một số tài nguyên Rq. Thí dụ, trong hình VI-7 dưới đây, chúng ta trình bày đồ thị cấp phát tài nguyên và đồ thị chờ tương ứng.

Như đã đề cập trước đó, deadlock tồn tại trong hệ thống nếu và chỉ nếu đồ thị chờ chứa chu trình. Để phát hiện deadlock, hệ thống cần duy trì đồ thị chờ và định kỳ gọi giải thuật để tìm kiếm chu trình trong đồ thị.

Một giải thuật phát hiện chu trình trong đồ thị yêu cầu độ phức tạp n2 thao tác, ở đây n là số cạnh của đồ thị.

Nhiều thể hiện của một loại tài nguyên

Lược đồ đồ thị chờ không thể áp dụng đối với hệ thống cấp phát tài nguyên với nhiều thể hiện cho mỗi loại tài nguyên. Giải thuật phát hiện deadlock mà chúng ta mô tả sau đây có thể áp dụng cho hệ thống này. Giải thuật thực hiện nhiều cấu trúc dữ liệu thay đổi theo thời gian mà chúng tương tự như được dùng trong giải thuật Banker.

- Available: một vector có chiều dài m hiển thị số lượng tài nguyên sẵn có của mỗi loại.
- Allocation: ma trận nxm định nghĩa số lượng tài nguyên của mỗi loại hiện được cấp phát tới mỗi quá trình.
- Request: ma trận nxm hiển thị yêu cầu hiện tại của mỗi quá trình. Nếu Request[i,j] = k, thì quá trình Pi đang yêu cầu k thể hiện nữa của loại tài nguyên Rj.

Quan hệ   giữa hai vectors được định nghĩa trong phần VI.7.3. Để ký hiệu đơn giản, chúng ta sẽ xem những hàng trong ma trận Allocation và Request như các vector, và sẽ tham chiếu chúng như Allocationi và Requesti tương ứng. Giải thuật phát hiện được mô tả ở đây đơn giản khảo sát mọi thứ tự cấp phát có thể đối với các quá trình còn lại để được hoàn thành. So sánh giải thuật này với giải thuật Banker.

1. Gọi Work và Finish là các vector có chiều dài m và n tương ứng. Khởi tạo Work:=Available. Cho i = 1, 2, …,n, nếu Allocationi ≠ 0, thì Finish[i]:= false; ngược lại Finish[i]:= true.
2. Tìm chỉ số i thỏa:
3. Finish[i] = false
4. Request i   Work.

Nếu không có i nào thỏa, di chuyển tới bước 4

1. Work:=Work + Allocation i

Finish[i] := true

Di chuyển về bước 2.

1. Nếu Finish[i] = false cho một vài i, 1 i n thì hệ thống đang ở trạng thái deadlock. Ngoài ra, nếu Finish[i] = false thì quá trình Pi bị deadlock.

Giải thuật này yêu cầu độ phức tạp mxn2 để phát hiện hệ thống có ở trong trạng thái deadlock hay không.

Câu hỏi đặt ra là tại sao chúng ta trả lại tài nguyên của quá trình Pi (trong bước 3) ngay sau khi chúng ta xác định rằng Requesti Work (trong bước 2b). Chúng ta biết rằng Pi hiện tại không liên quan deadlock (vì Requesti Work). Do đó, chúng ta lạc quan và khẳng định rằng Pi sẽ không yêu cầu thêm tài nguyên nữa để hoàn thành công việc của nó; do đó nó sẽ trả về tất cả tài nguyên hiện được cấp phát tới hệ thống. Nếu giả định của chúng ta không đúng, deadlock có thể xảy ra sao đó. Deadlock sẽ được phát hiện tại thời điểm kế tiếp mà giải thuật phát hiện deadlock được nạp.

Để minh hoạ giải thuật này, chúng ta xét hệ thống với 5 quá trình P0 đến P4 và 3 loại tài nguyên A, B, C. Loại tài nguyên A có 7 thể hiện, loại tài nguyên B có 2 thể hiện và loại tài nguyên C có 6 thể hiện. Giả sử rằng tại thời điểm T0, chúng ta có trạng thái cấp phát tài nguyên sau:

	Allocation			Request			Available		
	A	B	C	A	B	C	A	B	C
P0	0	1	0	0	0	0	0	0	0
P1	2	0	0	2	0	2			
P2	3	0	3	0	0	0			
P3	2	1	1	1	0	0			
P4	0	0	2	0	0	2			

Chúng ta khẳng định rằng hệ thống không ở trong trạng thái deadlock. Thật vậy, nếu chúng ta thực thi giải thuật, chúng ta sẽ tìm ra thứ tự <P0, P2, P3, P1, P4> sẽ dẫn đến trong Finish[i] = true cho tất cả i.

Bây giờ giả sử rằng quá trình P2 thực hiện yêu cầu thêm thể hiện loại C. Ma trận yêu cầu được hiệu chỉnh như sau:

	Need		
	A	B	C

P0	0	0	0
P1	2	0	2
P2	0	0	1
P3	1	0	0
P4	0	0	2

Chúng ta khẳng định rằng hệ thống bây giờ bị deadlock. Mặc dù chúng ta có thể đòi lại tài nguyên được giữ bởi quá trình P0, số lượng tài nguyên sẵn dùng không đủ để hoàn thành các yêu cầu của các quá trình còn lại. Do đó, deadlock tồn tại và bao gồm các quá trình P1, P2, P3, P4.

Sử dụng giải thuật phát hiện deadlock

Khi nào thì chúng ta nên nạp giải thuật phát hiện deadlock? Câu trả lời phụ thuộc vào hai yếu tố:

1. Deadlock có khả năng xảy ra thường xuyên như thế nào?
2. Bao nhiêu quá trình sẽ bị ảnh hưởng bởi deadlock khi nó sẽ ra?

Nếu deadlock xảy ra thường xuyên thì giải thuật phát hiện nên được nạp lên thường xuyên. Những tài nguyên được cấp phát để các quá trình bị deadlock sẽ rảnh cho đến khi deadlock có thể bị phá vỡ. Ngoài ra, số lượng quá trình liên quan trong chu trình deadlock có thể tăng lên.

Deadlock xảy ra chỉ khi một số quá trình thực hiện yêu cầu mà không được cấp tài nguyên tức thì. Yêu cầu này có thể là yêu cầu cuối hoàn thành một chuỗi các quá trình đang yêu cầu. Ngoài ra, chúng ta có thể nạp giải thuật phát hiện mọi khi một yêu cầu cho việc cấp phát không thể được cấp tức thì. Trong trường hợp này, chúng ta không chỉ định nghĩa tập hợp các quá trình bị deadlock, mà còn xác định quá trình đã gây ra deadlock. (Trong thực

tế, mỗi quá trình trong suốt quá trình bị deadlock là một liên kết trong chu trình của đồ thị tài nguyên, vì thế tất cả chúng gây ra deadlock). Nếu có nhiều loại tài nguyên khác nhau, một yêu cầu có thể gây chu trình trong đồ thị tài nguyên, mỗi chu trình hoàn thành bởi yêu cầu mới nhất và "được gây ra" bởi một quá trình có thể xác định.

Dĩ nhiên, nạp giải thuật phát hiện deadlock cho mỗi yêu cầu có thể gây ra một chi phí có thể xem xét trong thời gian tính toán. Một thay đổi ít đắt hơn là nạp giải thuật tại thời điểm ít thường xuyên hơn- thí dụ, một lần một giờ hay bất cứ khi nào việc sử dụng CPU rơi xuống thấp hơn 40%. Nếu giải thuật phát hiện deadlock được nạp trong những thời điểm bất kỳ, thì có nhiều chu trình trong đồ thị tài nguyên. Chúng ta không thể nói quá trình nào của nhiều quá trình bị deadlock gây ra deadlock.

Phục hồi deadlock

Khi giải thuật phát hiện xác định rằng deadlock tồn tại, nhiều thay đổi tồn tại. Một khả năng là thông báo người điều hành rằng deadlock xảy ra và để người điều hành giải quyết deadlock bằng thủ công. Một khả năng khác là để hệ thống tự động phục hồi. Có hai tuỳ chọn cho việc phá vỡ deadlock. Một giải pháp đơn giản là huỷ bỏ một hay nhiều quá trình để phá vỡ việc tồn tại chu trình trong đồ thị cấp phát tài nguyên. Tuỳ chọn thứ hai là lấy lại một số tài nguyên từ một hay nhiều quá trình bị deadlock.

Kết thúc quá trình

Để xóa deadlock bằng cách hủy bỏ quá trình, chúng ta dùng một trong hai phương pháp. Trong cả hai phương pháp, hệ thống lấy lại tài nguyên được cấp phát đối với quá trình bị kết thúc.

- Huỷ bỏ tất cả quá trình bị deadlock: phương pháp này rõ ràng sẽ phá vỡ chu trình deadlock, nhưng chi phí cao; các quá trình này có thể đã tính toán trong thời gian dài, và các kết quả của các tính toán từng phần này phải bị bỏ đi và có thể phải tính lại sau đó.

- Hủy bỏ một quá trình tại thời điểm cho đến khi chu trình deadlock bị xóa:phương pháp này chịu chi phí có thể xem xét vì sau khi mỗi quá trình bị hủy bỏ, một giải thuật phát hiện deadlock phải được nạp lên để xác định có quá trình nào vẫn đang bị deadlock.

Hủy bỏ quá trình có thể không dễ. Nếu một quá trình đang ở giữa giai đoạn cập nhật một tập tin, kết thúc nó sẽ để tập tin đó trong trạng thái không phù hợp. Tương tự, nếu quá trình đang ở giữa giai đoạn in dữ liệu ra máy in, hệ thống phải khởi động lại trạng thái đúng trước khi in công việc tiếp theo.

Nếu phương pháp kết thúc một phần được dùng thì với một tập hợp các quá trình deadlock được cho, chúng ta phải xác định quá trình nào (hay các quá trình nào) nên được kết thúc trong sự cố gắng để phá vỡ deadlock. Việc xác định này là một quyết định chính sách tương tự như các vấn đề lập thời biểu CPU. Câu hỏi về tính kinh tế là chúng ta nên hủy bỏ quá trình nào thì sự kết thúc của quá trình đó sẽ chịu chi phí tối thiểu. Tuy nhiên, thuật ngữ chi phí tối thiểu là không chính xác. Nhiều yếu tố có thể xác định quá trình nào được chọn bao gồm:

1. Độ ưu tiên của quá trình là gì.
2. Quá trình đã được tính toán bao lâu và bao lâu nữa quá trình sẽ tính toán trước khi hoàn thành tác vụ được chỉ định của nó.
3. Bao nhiêu và loại tài nguyên gì quá trình đang sử dụng.
4. Bao nhiêu tài nguyên nữa quá trình cần để hoàn thành
5. Bao nhiêu quá trình sẽ cần được kết thúc.
6. Quá trình là giao tiếp hay dạng bó

Lấy lại tài nguyên

Để xóa deadlock sử dụng việc trả lại tài nguyên, sau khi chúng ta đòi một số tài nguyên từ các quá trình và cho các tài nguyên này tới các quá trình khác cho đến khi chu trình deadlock bị phá vỡ.

Nếu việc đòi lại được yêu cầu để giải quyết deadlock thì ba vấn đề cần được xác định:

1. Chọn nạn nhân: những tài nguyên nào và những quá trình nào bị đòi lại? Trong khi kết thúc quá trình, chúng ta phải xác định thứ tự đòi lại để tối thiểu chi phí. Các yếu tố chi phí có thể gồm các tham số như số lượng tài nguyên một quá trình deadlock đang giữ, và lượng thời gian một quá trình deadlock dùng trong sự thực thi của nó.
2. Trở lại trạng thái trước deadlock: Nếu chúng ta đòi lại tài nguyên từ một quá trình, điều gì nên được thực hiện với quá trình đó? Rõ ràng, nó không thể tiếp tục việc thực thi bình thường; nó đang bị mất một số tài nguyên được yêu cầu. Chúng ta phải phục hồi quá trình tới trạng thái an toàn và khởi động lại từ trạng thái gần nhất trước đó.
3. Thông thường, rất khó để xác định trạng thái gì là an toàn vì thế giải pháp đơn giản nhất là phục hồi toàn bộ: hủy quá trình và sau đó khởi động lại nó. Tuy nhiên, hữu hiệu hơn để phục hồi quá trình chỉ đủ xa cần thiết để phá vỡ deadlock. Ngoài ra, phương pháp này yêu cầu hệ thống giữ nhiều thông tin hơn về trạng thái của tất cả các quá trình đang chạy.
4. Đói tài nguyên: chúng ta đảm bảo như thế nào việc đói tài nguyên không xảy ra? Nghĩa là, chúng ta có thể đảm bảo rằng tài nguyên sẽ không luôn bị đòi lại từ cùng một quá trình.
5. Trong một hệ thống việc chọn nạn nhân ở đâu dựa trên cơ sở các yếu tố chi phí, nó có thể xảy ra cùng quá trình luôn được chọn như là nạn nhân. Do đó, quá trình này không bao giờ hoàn thành tác vụ được chỉ định của nó, một trường hợp đói tài nguyên cần được giải quyết trong bất kỳ hệ thống thực tế. Rõ ràng, chúng ta phải đảm bảo một quá trình có thể được chọn như một nạn nhân chỉ một số lần xác định (nhỏ). Giải pháp chung nhất là bao gồm số lượng phục hồi trong yếu tố chi phí.

Tóm tắt

Trạng thái deadlock xảy ra khi hai hay nhiều quá trình đang chờ không xác định một sự kiện mà có thể được gây ra chỉ bởi một trong những quá trình đang chờ. Về nguyên tắc, có ba phương pháp giải quyết deadlock:

- Sử dụng một số giao thức để ngăn chặn hay tránh deadlock, đảm bảo rằng hệ thống sẽ không bao giờ đi vào trạng thái deadlock.

- Cho phép hệ thống đi vào trạng thái deadlock, phát hiện và sau đó phục hồi.
- Bỏ qua vấn đề deadlock và giả vờ deadlock chưa bao giờ xảy ra trong hệ thống. Giải pháp này là một giải pháp được dùng bởi hầu hết các hệ điều hành bao gồm UNIX.

Trường hợp deadlock có thể xảy ra nếu và chỉ nếu bốn điều kiện cần xảy ra cùng một lúc trong hệ thống: loại trừ hỗ tương, giữ và chờ cấp thêm tài nguyên, không đòi lại tài nguyên, và tồn tại chu trình trong đồ thị cấp phát tài nguyên. Để ngăn chặn deadlock, chúng ta đảm bảo rằng ít nhất một điều kiện cần không bao giờ xảy ra.

Một phương pháp để tránh deadlock mà ít nghiêm ngặt hơn giải thuật ngăn chặn deadlock là có thông tin trước về mỗi quá trình sẽ đang dùng tài nguyên như thế nào. Thí dụ, giải thuật Banker cần biết số lượng tối đa của mỗi lớp tài nguyên có thể được yêu cầu bởi mỗi quá trình. Sử dụng thông tin này chúng ta có thể định nghĩa giải thuật tránh deadlock.

Nếu hệ thống không thực hiện một giao thức để đảm bảo rằng deadlock sẽ không bao giờ xảy ra thì lược đồ phát hiện và phục hồi phải được thực hiện. Giải thuật phát hiện deadlock phải được nạp lên để xác định deadlock có thể xảy ra hay không. Nếu deadlock được phát hiện hệ thống phải phục hồi bằng cách kết thúc một số quá trình bị deadlock hay đòi lại tài nguyên từ một số quá trình bị deadlock.

Trong một hệ thống mà nó chọn các nạn nhân để phụv hồi về trạng thái trước đó chủ yếu dựa trên cơ sở yếu tố chi phí, việc đói tài nguyên có thể xảy ra. Kết quả là quá trình được chọn không bao giờ hoàn thành tác vụ được chỉ định của nó.

Quản lý bộ nhớ
1 Mục đích Sau khi học xong chương này, người học nắm được những kiến thức sau: - Hiểu các cách khác nhau để quản lý bộ nhớ - Hiểu tiếp cận quản lý bộ phân trang và phân đoạn - Vận dụng một tiếp cận quản lý bộ nhớ phù hợp với hệ thống xác định

Giới thiệu

Trong chương này chúng ta sẽ thảo luận nhiều cách khác nhau để quản lý bộ nhớ. Các giải thuật quản lý bộ nhớ từ tiếp cận máy trơ cơ bản (primitive bare-machine) là chiến lược phân trang và phân đoạn. Mỗi tiếp cận có lợi điểm và nhược của chính nó. Chọn phương pháp quản lý bộ nhớ cho một hệ thống xác định phụ thuộc vào nhiều yếu tố, đặc biệt trên thiết kế phần cứng của hệ thống. Chúng ta sẽ thấy nhiều giải thuật yêu cầu hỗ trợ phần cứng mặc dù các thiết kế gần đây đã tích hợp phần cứng và hệ điều hành.

Đặt vấn đề

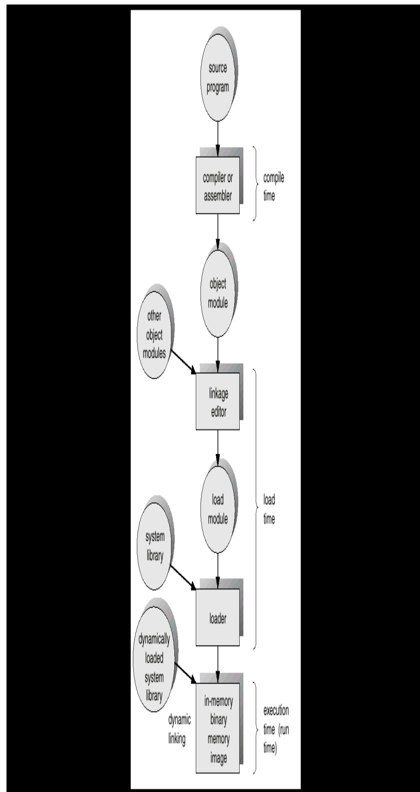
Bộ nhớ là trung tâm để điều hành hệ thống máy tính hiện đại. Bộ nhớ chứa một mảng lớn các từ (words) hay các bytes, mỗi phần tử với địa chỉ của chính nó. CPU lấy các chỉ thị từ bộ nhớ dựa theo giá trị của thanh đếm chương trình. Các chỉ thị này có thể gây việc nạp bổ sung các từ và lưu trữ tới các địa chỉ bộ nhớ xác định.

Liên kết địa chỉ

Thông thường, một chương trình nằm trên đĩa như một tập tin có thể thực thi dạng nhị phân. Chương trình này được mang vào trong bộ nhớ và được đặt trong một quá trình để nó được thực thi. Phụ thuộc vào việc quản lý bộ nhớ đang dùng, quá trình có thể được di chuyển giữa đĩa và bộ nhớ trong khi thực thi. Tập hợp các quá trình trên đĩa đang chờ được mang vào bộ nhớ để thực thi hình thành một hàng đợi nhập (input queue).

Thủ tục thông thường là chọn một trong những quá trình trong hàng đợi nhập và nạp quá trình đó vào trong bộ nhớ. Khi một quá trình được thực thi, nó truy xuất các chỉ thị và dữ liệu từ bộ nhớ. Cuối cùng, một quá trình kết thúc và không gian bộ nhớ của nó được xác định là trống.

Hầu hết các hệ thống cho phép một quá trình người dùng nằm ở bất cứ phần nào của bộ nhớ vật lý. Do đó, mặc dù không gian địa chỉ của máy tính bắt đầu tại 00000, nhưng địa chỉ đầu tiên của quá trình người dùng không cần tại 00000. Sắp xếp này ảnh hưởng đến địa chỉ mà chương trình người dùng có thể dùng. Trong hầu hết các trường hợp, một chương trình người dùng sẽ đi qua một số bước- một vài trong chúng có thể là tuỳ chọn-trước khi được thực thi (hình VII-1). Các địa chỉ có thể được hiện diện trong những cách khác trong những bước này. Các địa chỉ trong chương trình nguồn thường là những danh biểu. Một trình biên dịch sẽ liên kết các địa chỉ danh biểu tới các địa chỉ có thể tái định vị (chẳng hạn như 14 bytes từ vị trí bắt đầu của module này). Bộ soạn thảo liên kết hay bộ nạp sẽ liên kết các địa chỉ có thể tái định vị tới địa chỉ tuyệt đối (chẳng hạn 74014). Mỗi liên kết là một ánh xạ từ một không gian địa chỉ này tới một không gian địa chỉ khác.



Hình VII-1 Xử lý nhiều bước của chương trình người dùng

Về truyền thống, liên kết các chỉ thị và dữ liệu tới các địa chỉ có thể được thực hiện tại bất cứ bước nào theo cách sau đây:

- Thời gian biên dịch: nếu tại thời điểm biên dịch có thể biết quá trình nằm ở đâu trong bộ nhớ thì mã tuyệt đối có thể được phát sinh. Thí dụ, nếu biết trước quá trình người dùng nằm tại vị trí R thì mã trình biên dịch được phát sinh sẽ bắt đầu tại vị trí đó và mở rộng từ đó. Nếu tại thời điểm sau đó, vị trí bắt đầu thay đổi thì sẽ cần biên dịch lại mã này. Các chương trình định dạng .COM của MS-DOS là mã tuyệt đối giới hạn tại thời điểm biên dịch.
- Thời điểm nạp: nếu tại thời điểm biên dịch chưa biết nơi quá trình sẽ nằm ở đâu trong bộ nhớ thì trình biên dịch phải phát sinh mã có thể tái định vị. Trong trường hợp này, liên kết cuối cùng được trì hoãn cho tới thời điểm nạp. Nếu địa chỉ bắt đầu thay đổi, chúng ta chỉ cần nạp lại mã người dùng để hợp nhất giá trị được thay đổi này.
- Thời gian thực thi: nếu quá trình có thể được di chuyển trong thời gian thực thi từ một phân đoạn bộ nhớ này tới một phân đoạn bộ nhớ khác thì việc liên kết phải bị trì hoãn cho tới thời gian chạy. Phần cứng đặc biệt phải sẵn dùng cho cơ chế này để thực hiện công việc. Hầu hết những hệ điều hành này dùng phương pháp này.

Phần chủ yếu của chương này được dành hết để hiển thị các liên kết khác nhau có thể được cài đặt hữu hiệu trong một hệ thống máy tính và thảo luận sự hỗ trợ phần cứng tương ứng.

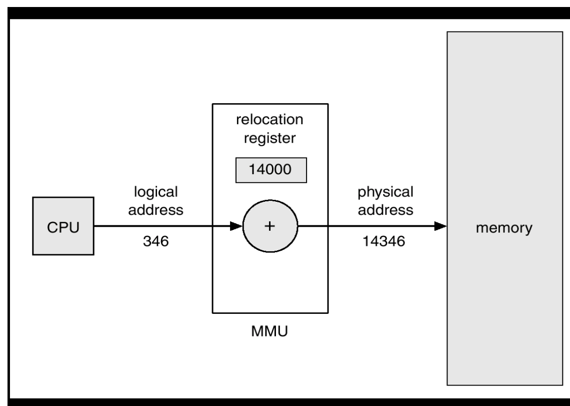
Không gian địa chỉ luận lý và không gian địa chỉ vật lý

Một địa chỉ được tạo ra bởi CPU thường được gọi là địa chỉ luận lý (logical address), ngược lại một địa chỉ được xem bởi đơn vị bộ nhớ-nghĩa là, một địa chỉ được nạp vào thanh ghi địa chỉ bộ nhớ-thường được gọi là địa chỉ vật lý (physical address).

Các phương pháp liên kết địa chỉ thời điểm biên dịch và thời điểm nạp tạo ra địa chỉ luận lý và địa chỉ vật lý xác định. Tuy nhiên, cơ chế liên kết địa chỉ tại thời điểm thực thi dẫn đến sự khác nhau giữa địa chỉ luận lý và địa chỉ vật lý. Trong trường hợp này, chúng ta thường gọi địa chỉ luận lý như là địa chỉ ảo (virtual address). Tập hợp tất cả địa chỉ luận lý được tạo ra bởi chương trình là không gian địa chỉ luận lý ; tập hợp tất cả địa chỉ vật lý tương ứng địa chỉ luận lý này là không gian địa chỉ vật lý. Do đó, trong cơ chế liên kết địa chỉ tại thời điểm thực thi, không gian địa chỉ luận lý và không gian địa chỉ vật lý là khác nhau.

Việc ánh xạ tại thời điểm thực thi từ địa chỉ ảo tới địa chỉ vật lý được thực hiện bởi một thiết bị phần cứng được gọi là bộ quản lý bộ nhớ MMU (memory-management unit). Chúng ta có thể chọn giữa những phương pháp khác nhau để thực hiện việc ánh xạ.

Như được hiển thị trong hình VII-2 ở trên, phương pháp này yêu cầu sự hỗ trợ phần cứng. Thanh ghi nền bây giờ được gọi là thanh ghi tái định vị. Giá trị trong thanh ghi tái định vị được cộng vào mỗi địa chỉ được tạo ra bởi quá trình người dùng tại thời điểm nó được gởi tới bộ nhớ. Thí dụ, nếu giá trị nền là 14000, thì việc cố gắng bởi người dùng để xác định vị trí 0 được tự động tái định vị tới vị trí 14000; một truy xuất tới địa chỉ 346 được ánh xạ tới vị trí 14346.



Hình VII-2 định vị tự động dùng thanh ghi tái định vị

Nạp động

Trong thảo luận của chúng ta gần đây, toàn bộ chương trình và dữ liệu của một quá trình phải ở trong bộ nhớ vật lý để quá trình thực thi. Kích thước của quá trình bị giới hạn bởi kích thước của bộ nhớ vật lý. Để đạt được việc sử dụng không gian bộ nhớ tốt hơn, chúng ta có thể sử dụng nạp động (dynamic loading). Với nạp động, một thủ tục không được nạp cho tới khi nó được gọi. Tất cả thủ tục được giữ trên đĩa trong định dạng nạp có thể tái định vị. Chương trình chính được nạp vào bộ nhớ và được thực thi. Khi một thủ tục cần gọi một thủ tục khác, thủ tục gọi trước hết kiểm tra để thấy thủ tục khác được nạp hay không. Nếu không, bộ nạp liên kết có thể tái định vị được gọi để nạp thủ tục mong muốn vào bộ nhớ và cập nhật các bảng địa chỉ của chương trình để phản ánh thay đổi này. Sau đó, điều khiển này được truyền tới thủ tục mới được nạp.

Thuận lợi của nạp động là ở đó một thủ tục không được dùng thì không bao giờ được nạp. Phương pháp này đặc biệt có ích khi lượng lớn mã được yêu cầu quản lý các trường hợp xảy ra không thường xuyên, chẳng hạn như các thủ tục lỗi. Trong trường hợp này, mặc dù kích thước toàn bộ chương trình có thể lớn, nhưng phần được dùng (và do đó được nạp) có thể nhỏ hơn nhiều.

Nạp động không yêu cầu hỗ trợ đặc biệt từ hệ điều hành. Nhiệm vụ của người dùng là thiết kế các chương trình của họ để đạt được sự thuận lợi đó. Tuy nhiên, hệ điều hành có thể giúp người lập trình bằng cách cung cấp các thủ tục thư viện để cài đặt nạp tự động.

Liên kết động và các thư viện được chia sẻ

Trong hình VII-1 cũng hiển thị thư viện được liên kết động. Một số hệ điều hành hỗ trợ chỉ liên kết tĩnh mà trong đó các thư viện ngôn ngữ hệ thống được đối xử như bất kỳ module đối tượng khác và được kết hợp bởi bộ nạp thành hình ảnh chương trình nhị phân. Khái niệm liên kết động là tương tự như khái niệm nạp động. Liên kết bị trì hoãn hơn là việc nạp bị trì hoãn cho tới thời điểm thực thi. Đặc điểm này thường được dùng với các thư viện hệ thống như các thư viện chương trình con của các ngôn ngữ. Không có tiện ích này, tất cả chương trình trên một hệ thống cần có một bản sao thư viện của ngôn ngữ của chúng (hay ít nhất thư viện được tham chiếu bởi chương trình) được chứa trong hình ảnh có thể thực thi. Yêu cầu này làm lãng phí cả không gian đĩa và bộ nhớ chính. Với liên kết động, một stub là một đoạn mã hiển thị cách định vị chương trình con trong thư viện cư trú trong bộ nhớ hay cách nạp thư viện nếu chương trình con chưa hiện diện.

Khi stub này được thực thi, nó kiểm tra để thấy chương trình con được yêu cầu đã ở trong bộ nhớ hay chưa. Nếu chưa, chương trình này sẽ nạp chương trình con vào trong bộ nhớ. Dù là cách nào, stub thay thế chính nó với địa chỉ của chương trình con và thực thi chương trình con đó. Do đó, thời điểm tiếp theo phân đoạn mã đạt được, chương trình con trong thư viện được thực thi trực tiếp mà không gây ra bất kỳ chi phí cho việc liên kết động. Dưới cơ chế này, tất cả các quá trình sử dụng một thư viện ngôn ngữ thực thi chỉ một bản sao của mã thư viện.

Phủ lắp

Để cho phép một quá trình lớn hơn lượng bộ nhớ được cấp phát cho nó, chúng ta sử dụng cơ chế phủ lắp (overlays). Ý tưởng phủ lắp là giữ trong bộ nhớ những chỉ thị và dữ liệu được yêu cầu tại bất kỳ thời điểm nào được cho. Khi những chỉ thị đó được yêu cầu, chúng được nạp vào không gian được chiếm trước đó bởi các chỉ thị mà chúng không còn cần nữa.

Thí dụ, xét trình dịch hợp ngữ hai lần (two-pass assembler). Trong suốt lần thứ 1, nó xây dựng bảng danh biểu; sau đó, trong lần thứ 2, nó tạo ra mã máy. Chúng ta có thể phân chia trình dịch hợp ngữ thành mã lần 1, mã lần 2, bảng danh biểu, và những thủ tục hỗ trợ chung được dùng bởi lần 1 và lần 2. Giả sử kích thước của các thành phần này như sau:

Lần 170 KB

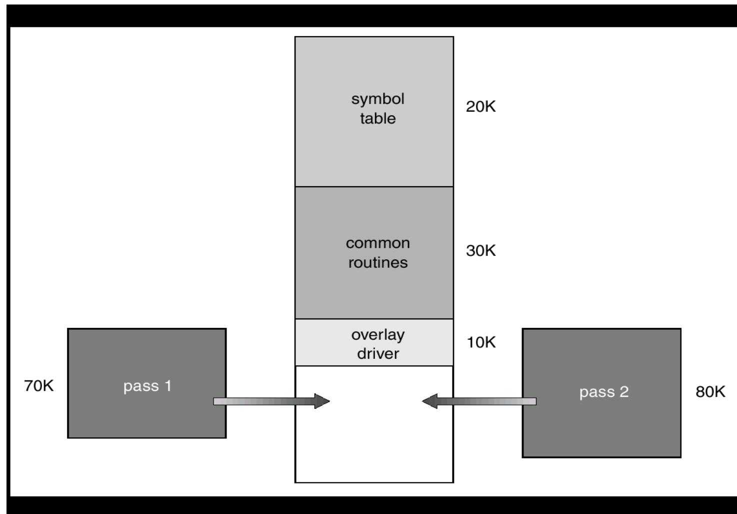
Lần 280 KB

Bảng danh biểu20 KB

Các thủ tục chung30 KB

Để nạp mọi thứ một lần, chúng ta cần 200KB bộ nhớ. Nếu chỉ có 150KB sẵn có, chúng ta không thể chạy quá trình của chúng ta. Tuy nhiên, chú ý rằng lần 1 và lần 2 không cần ở trong bộ nhớ cùng một lúc. Do đó, chúng ta định nghĩa hai phủ lắp. Phủ lắp A là bảng danh biểu, các thủ tục chung, lần 1, và phủ lắp B là bảng biểu tượng, các thủ tục chung và lần 2.

Chúng ta bổ sung trình điều khiển phủ lắp (10 KB) và bắt đầu với phủ lắp A trong bộ nhớ. Khi chúng ta kết thúc lần 1, chúng ta nhảy tới trình điều khiển phủ lắp, trình điều khiển này sẽ đọc phủ lắp B vào trong bộ nhớ, viết chồng lên phủ lắp B và sau đó chuyển điều khiển tới lần 2. Phủ lắp A chỉ cần 120KB, ngược lại phủ lắp B cần 130KB (hình VII-3). Bây giờ chúng ta có thể chạy trình hợp ngữ trong 150KB bộ nhớ. Nó sẽ nạp nhanh hơn vì rất ít dữ liệu cần được chuyển trước khi việc thực thi bắt đầu. Tuy nhiên, nó sẽ chạy chậm hơn do nhập/xuất phụ đọc mã mã cho phủ lắp A qua mã cho phủ lắp B.

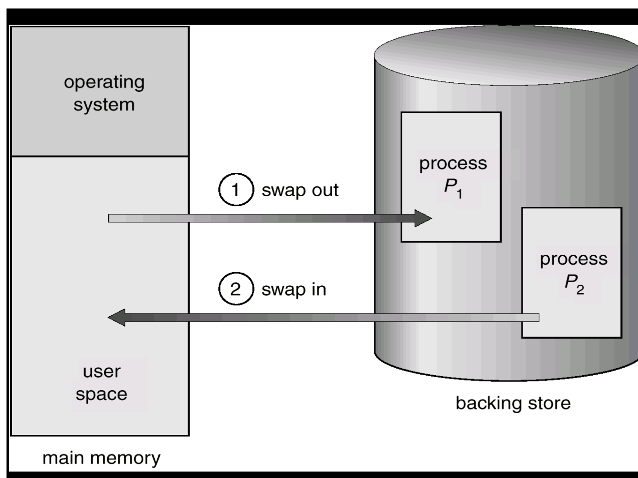


Hình VII-3- Các phủ lắp cho một bộ hợp ngữ dịch hai lần

Mã cho phủ lắp A và mã cho phủ lắp B được giữ trên đĩa như những hình ảnh bộ nhớ tuyệt đối, và được đọc bởi trình điều khiển phủ lắp khi cần. Tái định vị đặc biệt và các giải thuật liên kết được yêu cầu xây dựng các phủ lắp.

Hoán vị

Một quá trình cần ở trong bộ nhớ để được thực thi. Tuy nhiên, một quá trình có thể được hoán vị (swapped) tạm thời khỏi bộ nhớ tới vùng lưu trữ phụ backing store, sau đó mang trở lại bộ nhớ để việc thực thi được tiếp tục. Thí dụ, giả sử một môi trường đa chương với giải thuật lập thời biểu CPU round-robin. Khi định mức thời gian hết, bộ quản lý bộ nhớ sẽ bắt đầu hoán vị ra (swap out) vùng lưu trữ phụ quá trình vừa mới kết thúc và hoán vị vào (swap in) một quá trình khác tới không gian bộ nhớ được trống (hình VII-4). Do đó, bộ định thời biểu CPU sẽ cấp những phần thời gian tới những quá trình khác trong bộ nhớ. Lý tưởng, bộ quản lý sẽ hoán vị các quá trình đủ nhanh để một vài quá trình sẽ ở trong bộ nhớ, sẵn sàng thực thi, khi bộ định thời CPU muốn định thời lại CPU. Định mức cũng phải đủ lớn để phù hợp lượng tính toán được thực hiện giữa các hoán vị.



Hình VII-4- Hoán vị hai quá trình dùng đĩa như là backing store

Một biến thể của chính sách hoán vị này được dùng cho các giải thuật định thời trên cơ sở ưu tiên. Nếu một quá trình có độ ưu tiên cao hơn đến và muốn phục vụ, bộ quản lý bộ nhớ có thể hoán vị ra quá trình có độ ưu tiên thấp hơn để mà nó có thể nạp và thực thi quá trình có độ ưu tiên cao hơn. Khi quá trình có độ ưu tiên cao hơn kết thúc, quá trình có độ ưu tiên thấp hơn có thể được hoán vị vào trở lại và được tiếp tục. Biến thể của hoán vị này thường được gọi là cuộn ra (roll out), và cuộn vào (roll in).

Thông thường, một quá trình được hoán vị ra sẽ được hoán vị trở lại vào cùng không gian bộ nhớ mà nó đã chiếm trước đó. Sự giới hạn này được sai khiến bởi phương pháp liên kết địa chỉ. Nếu liên kết địa chỉ được thực hiện tại thời điểm hợp dịch hay nạp thì quá trình không thể được di chuyển vào không gian bộ nhớ khác vì các địa chỉ vật lý được tính trong thời gian thực thi.

Hoán vị yêu cầu một vùng lưu trữ phụ (backing store). Vùng lưu trữ phụ này thường là một đĩa tốc độ cao. Nó phải đủ lớn để chứa các bản sao của tất cả hình ảnh bộ nhớ cho tất cả người dùng, và nó phải cung cấp truy xuất trực tiếp tới các hình ảnh bộ nhớ này. Hệ thống này duy trì một hàng đợi sẵn sàng chứa tất cả quá trình mà các hình ảnh bộ nhớ của nó ở trong vùng lưu trữ phụ hay trong bộ nhớ và sẵn sàng để thực thi. Bất cứ khi nào bộ định thời CPU quyết định thực thi một quá trình, nó gọi bộ phân phát (dispacher). Bộ phân phát kiểm tra để thấy quá trình tiếp theo trong hàng đợi ở trong bộ nhớ không. Nếu không, và không có vùng bộ nhớ trống, bộ phân phát hoán vị ra một quá trình hiện hành trong bộ nhớ và hoán vị vào một quá trình mong muốn. Sau đó, nó nạp lại các thanh ghi và chuyển điều khiển tới quá trình được chọn.

Trong các hệ hoán vị, thời gian chuyển đổi giữa các tác vụ cần được quan tâm. Mỗi quá trình cần được phân chia một khoảng thời gian sử dụng CPU đủ lớn để không thấy rõ sự chậm trễ do các thao tác hoán vị gây ra. Nếu không, hệ thống sẽ dùng phần lớn thời gian để hoán vị các quá trình vào ra bộ nhớ chính, CPU như vậy sẽ không sử dụng hiệu quả.

Hoán vị cũng bị ràng buộc bởi yếu tố khác. Nếu chúng ta muốn hoán vị một quá trình, chúng ta phải đảm bảo rằng nó hoàn toàn rỗi. Quan tâm đặc biệt là việc chờ nhập/xuất. Một quá trình có thể đang chờ thao tác nhập/xuất khi chúng ta hoán vị quá trình đó tới nơi trống bộ nhớ của nó. Tuy nhiên, nếu nhập/xuất đang truy xuất không đồng bộ bộ nhớ người dùng cho nhập/xuất vùng đệm, thì quá trình không thể được hoán vị. Giả sử rằng thao tác nhập/xuất đang xếp hàng chờ vì thiết bị đang bận. Sau đó, nếu chúng ta hoán vị quá trình P1 ra và hoán vị P2 vào thì thao tác nhập/xuất có thể cố gắng sử dụng bộ nhớ hiện thuộc về quá trình P2. Hai giải pháp chủ yếu cho quá trình này là không bao giờ hoán vị quá trình đang chờ nhập/xuất hay thực thi các thao tác nhập/xuất chỉ ở trong vùng đệm của hệ điều hành. Chuyển giữa các vùng đệm của hệ điều hành và bộ nhớ quá trình thì chỉ xảy ra khi quá trình được hoán vị vào.

Cấp phát bộ nhớ liên tục

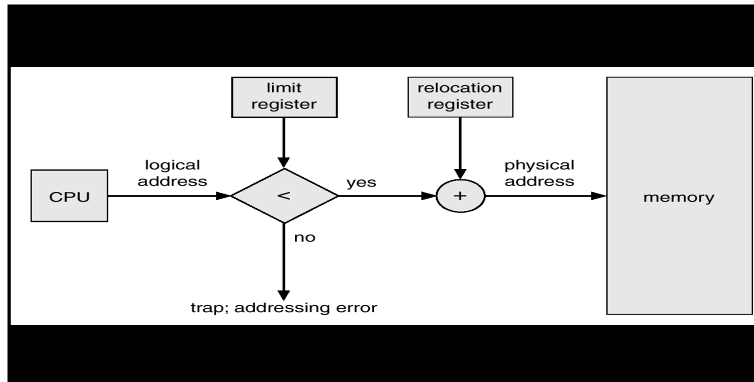
Bộ nhớ chính phải cung cấp cho cả hệ điều hành và các quá trình người dùng khác nhau. Do đó, chúng ta cần cấp phát những phần khác nhau của bộ nhớ chính trong những cách hiệu quả nhất có thể. Phần này chúng ta giải thích một phương pháp thông dụng, cấp phát bộ nhớ liên tục.

Bộ nhớ thường được phân chia thành hai phân khu, một cho hệ điều hành định vị và một cho các quá trình người dùng. Chúng ta có thể đặt hệ điều hành ở bộ nhớ cao hay bộ nhớ thấp. Yếu tố quan trọng ảnh hưởng tới quyết định này là vị trí của vector ngắt. Vì vector ngắt thường ở trong bộ nhớ thấp nên các lập trình viên thường cũng đặt hệ điều hành trong bộ nhớ thấp. Do đó, trong giáo trình này chúng ta sẽ thảo luận chỉ trường hợp hệ điều hành định vị trong bộ nhớ thấp. Phát triển của trường hợp khác là tương tự.

Chúng ta thường muốn nhiều quá trình người dùng định vị trong bộ nhớ tại cùng thời điểm. Do đó, chúng ta cần xem xét cách cấp phát bộ nhớ trống tới những quá trình ở trong hàng đợi nhập đang chờ được mang vào bộ nhớ. Trong cấp phát bộ nhớ liên tục, mỗi quá trình được chứa trong một phần bộ nhớ liên tục.

Bảo vệ bộ nhớ

Trước khi thảo luận cấp phát bộ nhớ chúng ta phải thảo luận vấn đề bảo vệ bộ nhớ-bảo vệ hệ điều hành từ quá trình người dùng, và bảo vệ các quá trình từ một quá trình khác. Chúng ta có thể cung cấp bảo vệ này bằng cách dùng thanh ghi tái định vị. Thanh ghi tái định vị chứa giá trị địa chỉ vật lý nhỏ nhất; thanh ghi giới hạn chứa dãy các định chỉ luận lý (thí dụ: tái định vị = 100040 và giới hạn = 74600). Với các thanh ghi tái định vị và giới hạn, mỗi địa chỉ luận lý phải ít hơn thanh ghi giới hạn; MMU ánh xạ địa chỉ luận lý động bằng cách cộng giá trị trong thanh ghi tái định vị. Địa chỉ được tái định vị này được gửi tới bộ nhớ (như hình VII-5).



Hình VII-5 Hỗ trợ phần cứng cho các thanh ghi tái định vị và các giới hạn

Khi bộ định thời CPU chọn một quá trình thực thi, bộ phân phát nạp thanh ghi tái định vị và giới hạn với các giá trị đúng như một phần của chuyển đổi ngữ cảnh. Vì mọi địa chỉ được phát sinh bởi CPU được kiểm tra dựa trên các thanh ghi này, chúng ta có thể bảo vệ hệ điều hành và các chương trình và dữ liệu người dùng khác từ việc sửa đổi bởi quá trình đang chạy này.

Cơ chế dùng thanh ghi tái định vị cung cấp một cách hiệu quả để cho phép kích thước hệ điều hành thay đổi động. Khả năng mềm dẽo này có thể mong muốn trong nhiều trường hợp. Thí dụ, hệ điều hành chứa mã và không gian vùng đệm cho trình điều khiển thiết bị. Nếu một trình điều khiển thiết bị (hay dịch vụ hệ điều hành khác) không được dùng phổ biến, nó không muốn giữ mã và dữ liệu trong bộ nhớ, khi chúng ta có thể dùng không gian đó cho mục đích khác. Những mã như thế thường được gọi là mã hệ điều hành tạm thời (transient operating system code); nó đến và đi khi được yêu cầu. Do đó, dùng mã này thay đổi kích thước của hệ điều hành trong khi thực thi chương trình.

Hệ thống đơn chương

Trong phương pháp này bộ nhớ được chia sẻ cho hệ điều hành và một chương trình duy nhất của người sử dụng. Tại một thời điểm, một phần của bộ nhớ sẽ do hệ điều hành chiếm giữ, phần còn lại thuộc về quá trình người dùng duy nhất trong hệ thống (Hình VII-6). Quá trình này được toàn quyền sử dụng bộ nhớ dành cho nó.

User processOperating system0xFFF…0

Hình VII-6 Tổ chức bộ nhớ trong hệ thống đơn chương

Khi bộ nhớ được tổ chức theo cách thức này, chỉ có thể xử lý một chương trình tại một thời điểm. Quan sát hoạt động của các quá trình, có thể nhận thấy rất nhiều tiến trình trải qua phần lớn thời gian để chờ các thao tác nhập/xuất hoàn thành. Trong suốt thời gian này, CPU ở trạng thái rỗi. Trong trường hợp như thế, hệ thống đơn chương không cho phép sử dụng hiệu quả CPU. Ngoài ra, sự đơn chương không cho phép nhiều người sử dụng làm việc đồng thời theo cơ chế tương tác. Để nâng cao hiệu suất sử dụng CPU, cần cho phép chế độ đa chương mà trong đó các quá trình chia sẻ CPU với nhau để hoạt động đồng hành.

Hệ thống đa chương với phân khu cố định

Một trong những phương pháp đơn giản nhất để cấp phát bộ nhớ là chia bộ nhớ thành những phân khu có kích thước cố định. Mỗi phân khu có thể chứa chính xác một quá trình. Do đó, cấp độ đa chương được giới hạn bởi số lượng phân khu. Trong phương pháp đa phân khu, khi một phân khu rảnh, một quá trình được chọn từ hàng đợi nhập và được nạp vào phân khu trống. Khi quá trình kết thúc, phân khu trở nên sẵn dùng cho một quá trình khác. Có hai tiếp cận để tổ chức hàng đợi:

- Sử dụng nhiều hàng đợi: mỗi phân khu sẽ có một hàng đợi tương ứng (hình VII-7a). Khi một quá trình mới được tạo ra, nó được đưa vào hàng đợi của phân khu có kích thước nhỏ nhất thoả nhu cầu chứa nó. Cách tổ chức này có khuyết điểm trong trường hợp các hàng đợi của một số phân khu trống trong khi các hàng đợi của các phân khu khác lại đầy, buộc các quá trình trong những hàng đợi này phải chờ được cấp phát bộ nhớ.
- Sử dụng một hàng đợi: tất cả các quá trình được đặt trong hàng đợi duy nhất (hình VII-7b). Khi có một phân khu trống, quá trình đầu tiên trong hàng đợi có kích thước phù hợp sẽ được đặt vào phân khu và cho xử lý.

400K600K200KPartition 3Partition 4Partition 1Operating systema. Sử dụng nhiều hàng đợi	Partition 1Partition 4Partition 3Operating systemb. Sử dụng một hàng đợi

Hình VII-7 Cấp phát phân khu có kích thước cố định

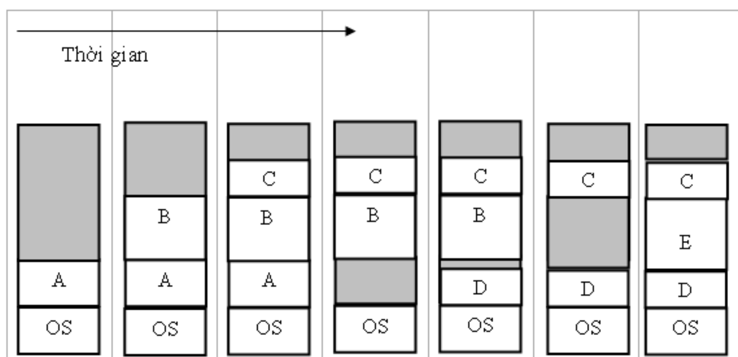
Khi sử dụng giải thuật này người ta muốn tránh sự hao phí một phân khu lớn cho một công việc nhỏ, nhưng lại xảy ra bất bình đẳng, bất lợi đối với các công việc nhỏ. Để giải quyết người ta thêm vào qui luật là một công việc sẽ không bị bỏ qua nữa nếu nó đã bị bỏ qua k lần qui định. Mỗi lần một công việc bị bỏ qua nó được đánh dấu một điểm. Khi đạt được số điểm qui định, nó sẽ không bị bỏ qua nữa, sẽ được nạp vào và thực hiện mặc dầu có thể trên một phân khu lớn hơn.

Phương pháp này ban đầu được sử dụng bởi hệ điều hành IBM OS/360, nó được gọi là MFT (Multiprogramming with Fixed number of Tasks). Hiện nay nó không còn sử dụng nữa.

Hệ thống đa chương với phân khu động

Cơ chế này là tổng quát của cơ chế phân khu cố định. Nó được dùng chủ yếu trong môi trường xử lý theo lô. Nhiều ý tưởng được trình bày ở đây cũng có thể áp dụng tới môi trường chia thời mà trong đó phân đoạn thuần được dùng cho việc quản lý bộ nhớ.

Hệ điều hành giữ một bảng hiển thị những phần nào của bộ nhớ là sẵn dùng và phần nào đang bận. Ban đầu, tất cả bộ nhớ là sẵn dùng cho quá trình người dùng, và được xem như một khối lớn bộ nhớ sẵn dùng hay một lỗ. Khi một quá trình đến và cần bộ nhớ, chúng ta tìm kiếm một lỗ trống đủ lớn cho quá trình này. Nếu chúng ta tìm thấy, chúng ta cấp phát chỉ phần bộ nhớ nhiều bằng lượng được yêu cầu, phần còn lại sẵn dùng để thoả mãn những yêu cầu tương lai (Hình VII-8).



Hình VII-8 Cấp phát các phân khu có kích thước thay đổi

Khi các quá trình đi vào hệ thống, chúng được đặt vào hàng đợi nhập. Hệ điều hành xem xét yêu cầu bộ nhớ của mỗi quá trình và lượng không gian bộ nhớ sẵn có để xác định các quá trình nào được cấp phát bộ nhớ. Khi một quá trình được cấp không gian, nó được nạp vào bộ nhớ và sau đó nó có thể cạnh tranh CPU. Khi một quá trình kết thúc, nó giải phóng bộ nhớ của nó, sau đó hệ điều hành có thể đặt một quá trình khác từ hàng đợi nhập.

Tại bất cứ thời điểm được cho, chúng ta có một danh sách kích thước khối trống và hàng đợi nhập. Hệ điều hành có thể xếp hàng đợi nhập dựa theo giải thuật định thời. Bộ nhớ được cấp phát tới quá trình cho đến khi các yêu cầu bộ nhớ của quá trình kế tiếp không thể được thoả; không có khối bộ nhớ trống (hay lỗ) đủ lớn để quản lý quá trình đó. Sau đó, hệ điều hành có thể chờ cho đến khi khối đủ lớn sẵn dùng hay nó có thể di chuyển xuống hàng đợi nhập để xem các yêu cầu bộ nhớ nhỏ hơn của các quá trình khác có thể được thoả hay không.

Thông thường, một tập hợp các lỗ có kích thước khác nhau được phân tán khắp bộ nhớ tại bất cứ thời điểm được cho. Khi một quá trình đến và yêu cầu bộ nhớ, hệ thống tìm tập hợp này một lỗ trống đủ lớn cho quá trình này. Nếu lỗ trống quá lớn, nó được chia làm hai: một phần được cấp tới quá trình đến; phần còn lại được trả về tập hợp các lỗ. Nếu lỗ mới nằm kề với các lỗ khác, các lỗ nằm kề này được gom lại để tạo thành một lỗ lớn hơn. Tại thời điểm này, hệ thống cần kiểm tra có quá trình nào đang chờ bộ nhớ và bộ nhớ trống mới hay bộ nhớ vừa được kết hợp lại có thể thoả yêu cầu của bất cứ quá trình đang chờ này không.

Thủ tục này là một trường hợp đặc biệt của vấn đề cấp phát lưu trữ động là làm cách nào để thoả mãn một yêu cầu có kích thước n từ danh sách lỗ trống. Có hai giải pháp chủ yếu cho vấn đề này.

1. Quản lý bằng bản đồ bit: bộ nhớ được chia thành các đơn vị cấp phát, mỗi đơn vị được ánh xạ tới một bit trong bản đồ bit (Hình VII-9). Giá trị bit này xác định trạng thái của đơn vị bộ nhớ đó: 0 đang tự do, 1 đã được cấp phát. Khi cần nạp một quá trình có kích thước k đơn vị, hệ thống sẽ tìm trong bản đồ bit một dãy k bit có giá trị 0.

Kích thước của đơn vị cấp phát là vấn đề lớn trong thiết kế. Nếu kích thước đơn vị cấp phát nhỏ sẽ làm tăng kích thước của bản đồ bit. Ngược lại, nếu kích thước đơn vị cấp phát lớn có thể gây hao phí cho đơn vị cấp phát sau cùng. Đây là giải pháp đơn giản nhưng thực hiện chậm nên ít được dùng.

Bản đồ bit tương ứngBộ nhớ có 5 quá trình và 3 lỗ trống ***SORRY, THIS MEDIA TYPE IS NOT SUPPORTED.***

Hình VII-8 Quản lý bộ nhớ bằng bản đồ bit

1. Quản lý bằng danh sách liên kết: dùng một danh sách liên kết để quản lý các phân đoạn bộ nhớ đã cấp phát và phân đoạn tự do, một phân đoạn có thể là một quá trình hay một vùng nhớ trống giữa hai quá trình. Danh sách liên kết gồm nhiều mục từ liên tiếp. Mỗi mục từ gồm 1 bit đầu để xác định phân đoạn đó là lỗ trống (H) hay một quá trình (P), sau đó là 3 từ để chỉ địa chỉ bắt đầu, chiều dài và chỉ điểm tới mục kế

tiếp. Việc sắp xếp các phân đoạn theo địa chỉ hay theo kích thước tuỳ thuộc vào giải thuật quản lý bộ nhớ. Sơ đồ quản lý bằng danh sách liên kết tương ứng với sơ đồ quản lý bằng bản đồ bit được minh hoạ trong hình VII-10.

SORRY, THIS MEDIA TYPE IS NOT SUPPORTED. Hình VII-9 Quản lý bộ nhớ bằng danh sách liên kết

Tập hợp các lỗ trống được tìm thấy để xác định lỗ nào là tốt nhất để cấp phát. Các chiến lược first-fit, best-fit, worst-fit là những chiến lược phổ biến nhất được dùng để chọn một lỗ trống từ tập hợp các lỗ trống.

- First-fit: cấp phát lỗ trống đầu tiên đủ lớn. Tìm kiếm có thể bắt đầu tại đầu tập hợp các lỗ trống hay tại điểm kết thúc của tìm kiếm first-fit trước đó. Chúng ta dừng tìm kiếm ngay khi chúng ta tìm thấy một lỗ trống đủ lớn.
- Best-fit: cấp phát lỗ trống nhỏ nhất đủ lớn. Chúng ta phải tìm toàn bộ danh sách, trừ khi danh sách được xếp thứ tự theo kích thước. Chiến lược này tạo ra lỗ trống nhỏ nhất còn thừa lại.
- Worst-fit: cấp phát lỗ trống lớn nhất. Chúng ta phải tìm toàn danh sách trừ khi nó được xếp theo thứ tự kích thước. Chiến lược này tạo ra lỗ trống còn lại lớn nhất mà có thể có ích hơn lỗ trống nhỏ từ tiếp cận best-fit.

Các mô phỏng hiển thị rằng cả first-fit và best-fit là tốt hơn worst-fit về việc giảm thời gian và sử dụng lưu trữ. Giữa first-fit và best-fit không thể xác định rõ chiến lược nào tốt hơn về sử dụng lưu trữ, nhưng first-fit có tốc độ nhanh hơn.

Tuy nhiên, các giải thuật này gặp phải vấn đề phân mãnh ngoài (external fragmentation). Khi các quá trình được nạp và được xoá khỏi bộ nhớ, không gian bộ nhớ trống bị phân rã thành những mãnh nhỏ. Phân mãnh ngoài tồn tại khi tổng không gian bộ nhớ đủ để thoả mãn một yêu cầu, nhưng nó không liên tục; vùng lưu trữ bị phân mãnh thành một số lượng lớn các lỗ nhỏ. Vấn đề phân mãnh này có thể rất lớn. Trong trường hợp xấu nhất, chúng có thể có một khối bộ nhớ trống nằm giữa mỗi hai quá trình. Nếu tất cả bộ nhớ này nằm trong một khối trống lớn, chúng ta có thể chạy nhiều quá trình hơn.

Chọn lựa first-fit so với best-fit có thể ảnh hưởng tới lượng phân mãnh. (First-fit là tốt hơn đối với một số hệ thống, ngược lại best fit là tốt hơn cho một số hệ thống khác). Một yếu tố khác là phần cuối của khối trống nào được cấp phát. (phần còn dư nào-phần trên đỉnh, hay phần ở dưới đáy?). Vấn đề không do giải thuật nào được dùng mà do phân mãnh ngoài.

Quản lý bộ nhớ với hệ thống bạn thân

Như ta đã thấy trong phần trước, việc quản lý các lỗ hổng trên những bảng liệt kê được sắp xếp theo kích thước làm cho việc cấp phát bộ nhớ rất nhanh, nhưng lại làm chậm cho việc ngưng cấp phát bởi vì ta phải chú ý đến các láng giềng. Hệ thống bạn thân (Buddy System) là một giải thuật quản lý bộ nhớ tận dụng thuận lợi của việc máy tính dùng những số nhị phân cho việc đánh địa chỉ để tăng tốc độ kết hợp những lỗ hổng sát nhau khi một quá trình hoàn thành hoặc được hoán vị ra ngoài.

Với phương pháp này, bộ quản lý bộ nhớ sẽ có một bảng liệt kê những khối còn tự do có kích thước 1, 2, 4, 16...bytes đến kích thước của bộ nhớ, tức là có kích thước bằng lũy thừa của 2. Khi có một quá trình cần cấp phát bộ nhớ, một lỗ hổng có kích thước bằng luỹ thừa của 2 đủ chứa quá trình sẽ được cấp phát. Nếu không có lỗ hổng yêu cầu, các lỗ hổng sẽ được phân đôi cho đến khi có được lỗ hỗng cần thiết. Khi một quá trình chấm dứt, các lỗ hổng kế nhau có kích thước bằng nhau sẽ được nhập lại để tạo thành lỗ hổng lớn hơn. Do đó, giải thuật này được gọi là hệ thống bạn thân.

Thí du: với bộ nhớ 1M, cần phải có 21 bảng liệt kê như thế sắp từ 1 bytes (20) đến 1 byte (220). Khởi đầu toàn bộ bộ nhớ còn tự do và bảng liệt kê 1M có một mục từ độc nhất chứa đựng một lỗ hổng 1M, tất cả các bảng liệt kê khác đều rỗng. Cấu hình bộ nhớ lúc khởi đầu được chỉ ra trong hình VII-11.

Memory

	0	128 K		256 K	384 K	512K	640 K	768 K	896 K	1M
Initially										1 Hole
request 70	A	128		256		512				3
request 35	A	B	64	256		512				3
request 80	A	B	64	C	128	512				3
return A	128	B	64	C	128	512				4
request 60	128	B	D	C	128	512				4
return B	128	64	D	C	128	512				4
return D	256			C	128	512				3
return C	1024									1

Hình VII-10Hệ thống bạn thân với kích thước 1M

Bây giờ chúng ta hãy xem cách hệ thống buddy làm việc khi một quá trình 70K được hoán vị vào bộ nhớ trống 1M. Do những lỗ hổng chỉ có thể có kích thước là lũy thừa của 2, 128K sẽ được yêu cầu, bởi vì đó chính là lũy thừa nhỏ nhất của 2 đủ lớn. Không có khối 128K sẵn, cũng không có các khối 256K và 512K. Vì vậy khối 1M sẽ được chia làm hai khối 512K, được gọi là những bạn thân; một tại địa chỉ 0 và một tại địa chỉ 512K. Sau đó khối tại địa chỉ thấp hơn, chính là khối tại 0 lại được phân làm hai khối bạn thân 256K, một tại 0 và một tại 256K. Cái thấp hơn của chúng lại được phân làm hai khối 128K, và khối tại 0, đánh dấu là A trong hình được cấp phát cho quá trình.

Kế đến, một quá trình 35K được hoán vị vào. Khi đó ta cần khối 64K, nhưng cũng không có sẵn, vì thế phải phân phối khối 128K thành hai khối bạn thân 64K, một tại địa chỉ 128K, một tại 192K. Khối tại 128K được cấp cho quá trình, trong hình là B. Yêu cầu thứ ba là 80K.

Bây giờ ta hãy xem những gì xảy ra khi một khối được trả lại. Giả sử tại thời điểm này khối 128K (mà chỉ dùng có 70K) được tự do. Khi đó khối 128K sẽ được đưa vào bảng tự do. Bây giờ cần một khối 60K. Sau khi kiểm tra, khối 64K tại 192K được cấp phát và nó được đánh dấu là C.

Bây giờ khối B được trả lại. Tại thời điểm này có hai khối 128K tự do nhưng chúng không được kết hợp lại. Chú ý rằng ngay cả khi khối 128K tại 0 được phân ra làm 2, khối tại 9 được dùng và khối tại 84K còn tự do, sự kết hợp cũng không xãy ra. Khi D được trả lại, sẽ có sự kết hợp lại thành khối 256K tại 0. Cuối cùng, khi C được trả lại, sẽ có kết hợp tạo thành 1 lỗ hổng 1M như ban đầu.

Hệ thống bạn thân có sự thuận lợi so với những giải thuật cùng sắp xếp theo kích thước của khối. Sự thuận lợi này là khi có một khối 2k bytes tự do, bộ quản lý bộ nhớ chỉ cần tìm trong bảng liệt kê lỗ hổng có kích thước 2k để xem chúng có khả năng kết hợp được hay không. Với những giải thuật khác mà trong đó cho phép các khối bộ nhớ được phân chia một cách tùy ý, việc tìm kiếm phải diễn ra trên tất cả các bảng liệt kê. Do đó, hệ thống bạn thân làm việc nhanh hơn.

Đáng tiếc, nó lại cực kỳ kém hiệu quả trong việc sử dụng bộ nhớ. Một quá trình 35K phải được cấp phát đến 64K, hao phí đến 29K. Sự hao phí này được gọi là sự phân mảnh trong (internal fragmentation), bởi vì phần bộ nhớ hao phí nằm bên trong đoạn được cấp phát. Còn trong các phần trước ta thấy những lỗ hổng ở giữa các đoạn, nhưng không có sự hao phí bên trong các đoạn, do đó kiểu này được coi là sự phân mảnh ngoài.

Phân mãnh

Phân mãnh bộ nhớ có thể là phân mãnh trong hoặc phân mãnh ngoài. Xét cơ chế cấp phát nhiều phân khu với một lỗ trống có kích thước 18,464 bytes. Giả sử rằng quá trình tiếp theo yêu cầu 18,462 bytes. Nếu chúng ta cấp phát chính xác khối được yêu cầu, chúng ta để lại một lỗ trống có kích thước 2 bytes. Chi phí để giữ vết của lỗ này sẽ lớn hơn kích thước của lỗ trống. Tiếp cận thông thường là chia bộ nhớ vật lý thành những khối có kích thước cố định, và cấp phát bộ nhớ dựa theo đơn vị của kích thước khối. Với tiếp cận này, bộ nhớ được cấp phát tới một quá trình có thể là lớn hơn một chút so với khối được yêu cầu. Sự chênh lệnh giữa hai số này là phân mãnh trong-bộ nhớ bị phân mãnh trong đối với một phân khu thì không thể được dùng.

Một giải pháp đối với phân mãnh ngoài là kết lại thành khối (compaction). Mục tiêu là di chuyển nội dung bộ nhớ để đặt tất cả bộ nhớ trống với nhau trong một khối lớn. Kết khối không phải luôn thực hiện được. Nếu việc tái định vị là tĩnh và được thực hiện tại thời điểm hợp dịch và nạp thì việc kết khối là không thể thực hiện được. Kết khối chỉ có thể thực hiện được chỉ nếu tái định vị là động và được thực hiện tại thời điểm thực thi. Nếu địa chỉ được tái định vị động, tái định vị yêu cầu chỉ di chuyển chương trình và dữ liệu, sau đó thay đổi thanh ghi nền để phản ánh địa chỉ nền mới. Khi kết khối là có thể, chúng ta phải xác định chi phí của nó. Giải thuật kết khối đơn giản nhất là di chuyển tất cả quá trình tới cuối bộ nhớ; tất cả lỗ trống di chuyển theo hướng ngược lại, tạo ra một lỗ trống lớn của bộ nhớ sẵn dùng. Cơ chế này có thể đắt.

Một giải pháp khác cho vấn đề phân mãnh ngoài là cho phép không gian địa chỉ luận lý của một quá trình là không liên tục, do đó cho phép một quá trình được cấp phát bộ nhớ vật lý bất cứ đâu sau khi sẵn dùng. Hai kỹ thuật bù trừ để đạt giải pháp này là phân trang và phân đoạn.

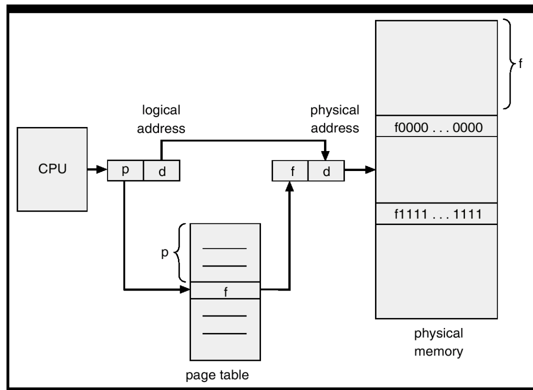
Cấp phát không liên tục

Phân trang

Phân trang là cơ chế quản lý bộ nhớ cho phép không gian địa chỉ vật lý của quá trình là không kề nhau. Phân trang tránh vấn đề đặt vừa khít nhóm bộ nhớ có kích thước thay đổi vào vùng lưu trữ phụ (backing store) mà hầu hết các cơ chế quản lý bộ nhớ trước đó gặp phải. Khi phân đoạn mã và dữ liệu nằm trong bộ nhớ được hoán vị ra, không gian phải được tìm thấy trên vùng lưu trữ phụ. Vấn đề phân mãnh được thảo luận trong sự kết nối với bộ nhớ chính cũng thông dụng như với vùng lưu trữ phụ, ngoại trừ truy xuất thấp hơn nhiều, vì thế kết khối là không thể. Vì lợi điểm của nó so với các phương pháp trước đó nên phân trang trong những dạng khác nhau được dùng phổ biến trong hầu hết các hệ điều hành.

Về truyền thống, hỗ trợ phân trang được quản lý bởi phần cứng. Tuy nhiên, những thiết kế gần đây cài đặt phân trang bằng cách tích hợp chặt chẻ phần cứng và hệ điều hành, đặc biệt trên các bộ vi xử lý 64-bit.

Phương pháp cơ bản



Hình VII-11 Phần cứng phân trang

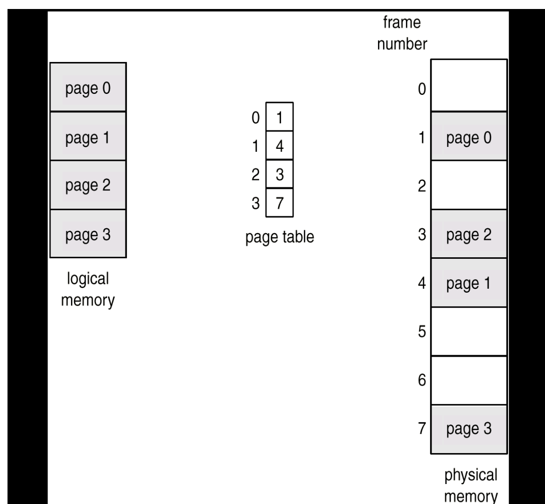
Bộ nhớ vật lý được chia thành các khối có kích thước cố định được gọi là các khung (frames). Bộ nhớ luận lý cũng được chia thành các khối có cùng kích thước được gọi là các trang (pages). Khi một quá trình được thực thi, các trang của nó được nạp vào các khung bộ nhớ sẵn dùng từ vùng lưu trữ phụ. Vùng lưu trữ phụ được chia thành các khối có kích thước cố định và có cùng kích thước như các khung bộ nhớ.

Hỗ trợ phần cứng cho phân trang được hiển thị trong hình VII-12. Mỗi địa chỉ được tạo ra bởi CPU được chia thành hai phần: số trang (p) và độ dời trang (d). Số trang được dùng như chỉ mục vào bảng trang. Bảng trang chứa địa chỉ nền của mỗi trang trong bộ nhớ vật lý. Địa chỉ nền này được kết hợp với độ dời trang để định nghĩa địa chỉ bộ nhớ vật lý mà nó được gởi đến đơn vị bộ nhớ. Mô hình phân trang bộ nhớ được hiển thị như hình VII-13.

Kích thước trang (giống như kích thước khung) được định nghĩa bởi phần cứng. Kích thước của một trang điển hình là luỹ thừa của 2, từ 512 bytes đến 16MB trên trang, phụ thuộc vào kiến trúc máy tính. Chọn luỹ thừa 2 cho kích thước trang thực hiện việc dịch địa chỉ luận lý thành số trang và độ dời trang rất dễ dàng. Nếu kích thước không gian địa chỉ là 2m, và kích thước trang là 2n đơn vị địa chỉ (byte hay từ) thì m-n bits cao của địa chỉ luận lý chỉ số trang, n bits thấp chỉ độ dời trang. Do đó, địa chỉ luận lý như sau:

Số trang	Độ dời trang
P	D
m –n	N

ở đây p là chỉ mục trong bảng trang và d là độ dời trong trang.



Hình VII-12 Mô hình phân trang của bộ nhớ luận lý và vật lý

Thí dụ: xét bộ nhớ trong hình VII-14. Sử dụng kích thước trang 4 bytes và bộ nhớ vật lý 32 bytes (có 8 trang), chúng ta hiển thị cách nhìn bộ nhớ của người dùng có thể được ánh xạ tới bộ nhớ vật lý như thế nào. Địa chỉ luận lý 0 là trang 0, độ dời 0. Chỉ mục trong bảng trang, chúng ta thấy rằng trang 0 ở trong khung 5. Do đó, địa chỉ luận lý 0 ánh xạ tới địa chỉ vật lý 20 (=(5x4)+0). Địa chỉ luận lý 3 (trang 0, độ dời 3) ánh xạ tới địa chỉ vật lý 23 (=(5x4)+3). Địa chỉ luận lý 4 ở trang 1, độ dời 0; dựa theo bảng trang, trang 1 được ánh xạ tới khung 6. Do đó, địa chỉ luận lý 4 ánh xạ tới địa chỉ 24(=(6x4)+0). Địa chỉ luận lý 13 ánh xạ tới địa chỉ vật lý 9.

Có thể chú ý rằng phân trang là một dạng của tái định vị động. Mỗi địa chỉ luận lý được giới hạn bởi phần cứng phân trang tới địa chỉ vật lý. Sử dụng phân trang tương tự sử dụng một bảng các thanh ghi nền (hay tái định vị), một thanh ghi cho mỗi khung bộ nhớ.

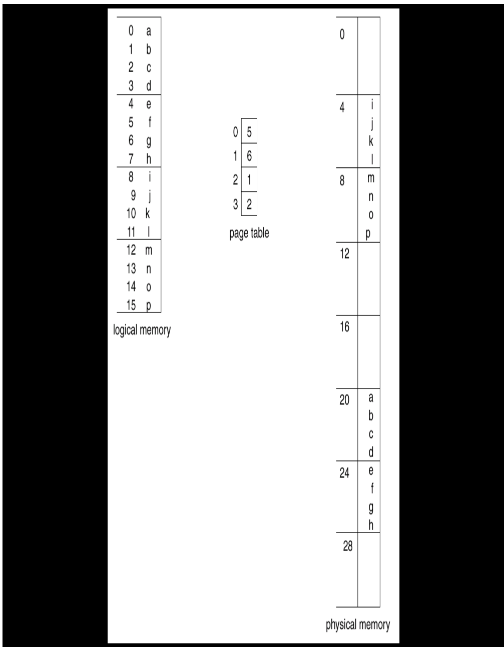
Khi chúng ta sử dụng một cơ chế phân trang, chúng ta không có phân mãnh bên ngoài: bất kỳ khung trống có thể được cấp phát tới một quá trình cần nó. Tuy nhiên, chúng ta có thể có phân mãnh trong. Chú ý rằng các khung được cấp phát như các đơn vị. Nếu các yêu cầu bộ nhớ của một quá trình không xảy ra để rơi trên giới hạn của trang, thì khung cuối cùng được cấp phát có thể không đầy hoàn toàn. Thí dụ, nếu các trang là 2048 bytes, một quá trình 72,766 bytes sẽ cần 35 trang cộng với 1086 bytes. Nó được cấp phát 36 khung, do đó phân mãnh trong là 2048 - 1086 = 962 bytes. Trong trường hợp xấu nhất, một quá trình cần n trang cộng với 1 byte. Nó sẽ được cấp phát n+1 khung, dẫn đến phân mãnh trong gần như toàn bộ khung.

Nếu kích thước quá trình độc lập với kích thước của trang, thì chúng ta mong muốn phân mãnh trong trung bình là ½ trang trên một quá trình. Xem xét này đề nghị rằng kích thước trang nhỏ là mong muốn. Tuy nhiên, chi phí liên quan tới mỗi mục từ bảng trang và chi phí này giảm khi kích thước trang tăng. Vì thế nhập/xuất đĩa là hiệu quả hơn khi số lượng dữ liệu được truyền lớn hơn. Thường thì kích thước trang lớn lên theo thời gian khi các quá trình, tập hợp dữ liệu, bộ nhớ chính trở nên lớn hơn. Ngày nay, các trang điển hình nằm trong khoảng 4 KB tới 8 KB, và một số hệ thống hỗ trợ kích thước trang lớn hơn. CPU và nhân thậm chí hỗ trợ nhiều kích thước khác nhau. Thí dụ, Solaris dùng 8 KB và 4 MB kích thước trang, phụ thuộc dữ liệu được lưu bởi trang. Hiện nay, các nhà nghiên cứu đang phát triển việc hỗ trợ kích thước trang khác nhau.

Mỗi mục từ bảng trang thường dài 4 bytes, nhưng kích thước có thể thay đổi. Một mục từ 32-bit có thể chỉ tới một khung trang vật lý 232. Nếu một khung là 4 KB, thì hệ thống với những mục từ 4 bytes có thể đánh địa chỉ cho 244 bytes (hay 16 TB) bộ nhớ vật lý.

Khi một quá trình đi vào hệ thống để được thực thi, kích thước của nó, được diễn tả trong các trang, được xem xét. Mỗi trang của quá trình cần trên một khung. Do đó, nếu quá trình yêu cầu n trang, ít nhất n khung phải sẵn dùng trong bộ nhớ. Nếu n khung là sẵn dùng, chúng được cấp phát tới quá trình đang đi vào này. Trang đầu tiên của quá trình được nạp vào một trong những khung được cấp phát, và số khung được đặt vào trong bảng trang

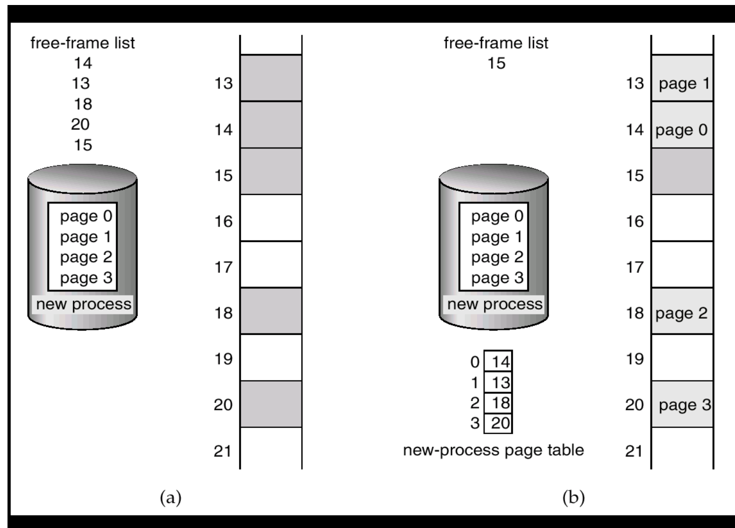
cho quá trình này. Trang kế tiếp được nạp vào một khung khác, và số khung của nó được đặt vào trong bảng trang, …(hình VII-15).



Hình VII-13 Thí dụ phân trang cho bộ nhớ 32 bytes với các trang có kích thức 4 bytes

Một khía cạnh quan trọng của phân trang là sự phân chia rõ ràng giữa tầm nhìn bộ nhớ của người dùng và bộ nhớ vật lý thật sự. Chương trình người dùng nhìn bộ nhớ như một không gian liên tục, chứa chỉ một chương trình. Sự thật, chương trình người dùng được phân bố khắp bộ nhớ vật lý mà nó cũng quản lý các quá trình khác. Sự khác nhau giữa tầm nhìn bộ nhớ của người dùng và bộ nhớ vật lý thật sự được làm cho tương thích bởi phần cứng dịch địa chỉ. Địa chỉ luận lý được dịch thành địa chỉ vật lý. Ánh xạ này được che giấu từ người dùng và được điều khiển bởi hệ điều hành. Chú ý rằng như định nghĩa, quá trình người dùng không thể truy xuất bộ nhớ mà nó không sở hữu. Không có cách định địa chỉ bộ nhớ bên ngoài bảng trang của nó và bảng chứa chỉ những trang mà quá trình sở hữu.

Vì hệ điều hành đang quản lý bộ nhớ vật lý nên nó phải hiểu những chi tiết cấp phát bộ nhớ vật lý; khung nào được cấp phát, khung nào còn trống, tổng khung hiện có là bao nhiêu,…Thông tin này được giữ trong một cấu trúc dữ liệu được gọi là bảng khung. Bảng khung chỉ có một mục từ cho mỗi khung trang vật lý, hiển thị khung trang đó đang rảnh hay được cấp phát. Nếu khung trang được cấp phát, thì xác định trang nào của quá trình nào được cấp.



Hình VII-14 các khung trống. (a) trước khi cấp phát. (b) sau khi cấp phát

Ngoài ra, hệ điều hành phải biết rằng quá trình người dùng hoạt động trong không gian người dùng, và tất cả địa chỉ luận lý phải được ánh xạ để phát sinh địa chỉ vật lý. Nếu người dùng thực hiện lời gọi hệ thống (thí dụ: để thực hiện nhập/xuất) và cung cấp địa chỉ như một tham số (thí dụ: vùng đệm), địa chỉ đó phải được ánh xạ để sinh ra địa chỉ vật lý đúng. Hệ điều hành duy trì một bản sao của bảng trang cho mỗi quá trình, như nó duy trì bản sao của bộ đếm chỉ thị lệnh và nội dung thanh ghi. Bản sao này được dùng để dịch địa chỉ luận lý thành địa chỉ vật lý bất cứ khi nào hệ điều hành phải ánh xạ địa chỉ luận lý tới địa chỉ vật lý dạng thủ công. Nó cũng được dùng bởi bộ phân phát CPU để địa chỉ bảng trang phần cứng khi một quá trình được cấp phát CPU. Do đó, trang gia tăng thời gian chuyển đổi ngữ cảnh.

Hỗ trợ phần cứng

Mỗi hệ điều hành có phương pháp riêng để lưu trữ các bảng trang. Hầu hết đều cấp phát một bảng trang cho mỗi quá trình. Một con trỏ chỉ tới một bảng trang được lưu trữ với những giá trị thanh ghi thanh ghi khác nhau (giống như bộ đếm chỉ thị lệnh) trong khối điều khiển quá trình. Khi bộ phân phát được yêu cầu bắt đầu một quá trình, nó phải nạp lại các thanh ghi người dùng và định nghĩa các giá trị bảng trang phần cứng phù hợp từ bảng trang người dùng được lưu trữ.

Cài đặt phần cứng của bảng trang có thể được thực hiện trong nhiều cách. Cách đơn giản nhất, bảng trang được cài đặt như tập hợp các thanh ghi tận hiến. Các thanh ghi này nên được xây dựng với tính logic tốc độ rất cao để thực hiện việc dịch địa chỉ trang hiệu quả. Mọi truy xuất tới bộ nhớ phải kiểm tra kỹ lưỡng bảng đồ trang, vì vậy tính hiệu quả là vấn đề xem xét chủ yếu. Bộ phân phát CPU nạp lại các thanh ghi này chỉ khi nó nạp lại các thanh ghi khác. Dĩ nhiên, các chỉ thị để nạp hay hiệu chỉnh các thanh ghi bảng trang phải được cấp quyền để mà chỉ hệ điều hành có thể thay đổi bản đồ bộ nhớ. DEC PDP-11 là một thí dụ về kiến trúc như thế. Địa chỉ chứa 16 bits, và kích thước trang là 8 KB. Do đó, bảng trang chứa 8 mục từ mà chúng được giữ trong các thanh ghi nhanh.

Sử dụng các thanh ghi cho bảng trang chỉ phù hợp nếu bảng trang có kích thước nhỏ (thí dụ: 256 mục từ). Tuy nhiên, hầu hết các máy tính tương thời cho phép bảng trang rất lớn (thí dụ, 1 triệu mục từ). Đối với những máy này, việc sử dụng các thanh ghi nhanh để cài đặt bảng trang là không khả thi. Hay đúng hơn là, bảng trang được giữ trong bộ nhớ chính, và thanh ghi nền bảng trang (page-table base register-PTBR) chỉ tới thanh ghi bảng trang. Thay đổi các bảng trang yêu cầu thay đổi chỉ một thanh ghi, về căn bản cắt giảm thời gian chuyển ngữ cảnh.

Vấn đề với tiếp cận này là thời gian được yêu cầu để truy xuất vị trí bộ nhớ người dùng. Nếu chúng ta muốn truy xuất vị trí i, đầu tiên chúng ta phải xác định chỉ mục trong bảng trang, sử dụng giá trị trong độ dời PTBR

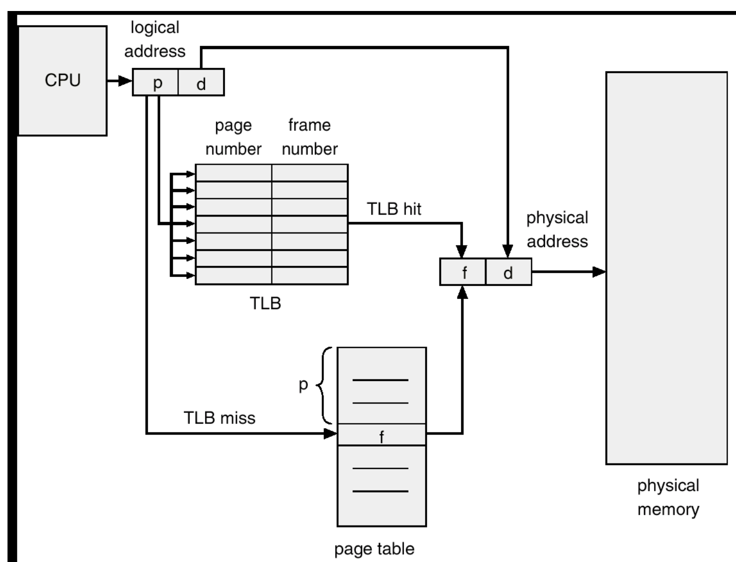
bởi số trang cho i. Tác vụ này yêu cầu một truy xuất bộ nhớ. Nó cung cấp chúng ta số khung được nối kết với độ dời trang để sinh ra địa chỉ thực. Sau đó, chúng ta có thể truy xuất tới nơi được mong muốn trong bộ nhớ. Với cơ chế này, hai truy xuất bộ nhớ được yêu cầu để truy xuất một byte (một cho mục từ bảng trang, một cho byte đó). Do đó, truy xuất bộ nhớ bị chậm bởi một trong hai yếu tố đó. Sự trì hoãn này không thể chấp nhận dưới hầu hết trường hợp vì thế chúng ta phải sử dụng đến hoán vị!

Giải pháp chuẩn cho vấn đề này là dùng bộ lưu trữ (cache) phần cứng đặc biệt, nhỏ, tìm kiếm nhanh được gọi là translation look-aside buffer (TLB). TLB là bộ nhớ kết hợp tốc độ cao. Mỗi mục từ trong TLB chứa hai phần: một khoá (key) và một giá trị (value). Khi bộ nhớ kết hợp được biểu diễn với một thành phần, nó được so sánh với tất cả khoá cùng một lúc. Nếu thành phần được tìm thấy, trường giá trị tương ứng được trả về. Tìm kiếm nhanh nhưng phần cứng đắt. Điển hình, số lượng mục từ trong TLB nhỏ, thường từ 64 đến 1024.

TLB được dùng với các bảng trang trong cách sau. TLB chứa chỉ một vài mục từ bảng trang. Khi một địa chỉ luận lý được phát sinh bởi CPU, số trang của nó được hiện diện trong TLB. Nếu số trang được tìm thấy, khung của nó lập tức sẵn dùng và được dùng để truy xuất bộ nhớ. Toàn bộ tác vụ có thể mất ít hơn 10% thời gian nếu dùng tham chiếu bộ nhớ không được ánh xạ.

Nếu số trang không ở trong TLB (còn gọi là mất TLB), tham chiếu bộ nhớ tới bảng trang phải được thực hiện. Khi số khung đạt được, chúng ta có thể dùng nó để truy xuất bộ nhớ (như hình VII-16). Ngoài ra, chúng ta thêm số trang và số khung tới TLB để mà chúng có thể được tìm thấy nhanh trên tham chiếu tiếp theo. Nếu một TLB đã đầy các mục từ, hệ điều hành phải chọn một mục từ để thay thế. Các chính sách thay thế rất đa dạng từ ít được sử dụng gần đây nhất (least recently used-LRU) tới chọn ngẫu nhiên. Ngoài ra, một số TLB cho phép các mục từ được wired down. Nghĩa là, chúng không thể được xoá khỏi TLB. Điển hình, các mục từ cho nhân thường được wired down.

Một số TLB lưu trữ các định danh không gian địa chỉ (address-space identifers-ASID) trong mỗi mục từ của TLB. Một ASID định danh duy nhất mỗi quá trình và được dùng để cung cấp việc bảo vệ không gian địa chỉ cho quá trình đó. Khi TLB cố gắng phân giải các số trang ảo, nó đảm bảo ASID cho mỗi quá trình hiện đang chạy trùng khớp với ASID được nối kết với trang ảo. Nếu các ASID không khớp, chúng được xem như mất TLB. Ngoài ra, để cung cấp việc bảo vệ không gian địa chỉ, một ASID cho phép TLB chứa các mục từ cho nhiều quá trình khác nhau cùng một lúc. Nếu TLB không hỗ trợ các ASID riêng thì mỗi lần một bảng trang được chọn (thí dụ, mỗi chuyển ngữ cảnh), một TLB phải được đẩy (hay được xoá) để đảm bảo rằng các quá trình đang thực thi tiếp theo không sử dụng thông tin dịch sai. Ngược lại, có những mục từ cũ trong TLB chứa các địa chỉ ảo nhưng có các địa chỉ không đúng hay không hợp lệ để lại từ quá trình trước.



Hình VII-15 phần cứng phân trang với TBL

Phần trăm thời gian mà số trang xác định được tìm thấy trong TLB được gọi là tỉ lệ chập (hit ratio). Tỉ lệ chập 80% có nghĩa là chúng ta tìm số trang mong muốn trong TLB 80% thời gian. Nếu mất 20 nano giây để tìm TLB và 100 nano giây để truy xuất bộ nhớ, thì một truy xuất bộ nhớ được ánh xạ mất 120 nano giây khi số trang ở trong TLB. Nếu chúng ta không tìm số trang trong TLB (20 nano giây) thì trước hết chúng ta phải truy xuất bộ nhớ cho bảng trang và số khung (100 nano giây), thì sau đó truy xuất byte mong muốn trong bộ nhớ (100 nano giây), tổng thời gian là 220 nano giây. Để tìm thời gian truy xuất bộ nhớ hiệu quả, chúng ta phải đo mỗi trường hợp với xác suất của nó:

Thời gian truy xuất hiệu quả = 0.80 x 120 + 0.2 x 220 = 140 nano giây

Trong thí dụ này, chúng ta gặp phải 40% chậm lại trong thời gian truy xuất bộ nhớ (từ 100 tới 140 nano giây).

Đối với một tỉ lệ chậm 98%, chúng ta có:

Thời gian truy xuất hiệu quả = 0.98 x 120 + 0.02 x 220 = 122 nano giây

Tỉ lệ chập được tăng này chỉ tạo ra 22% chậm lại trong thời gian truy xuất.

Sự bảo vệ

Bảo vệ bộ nhớ trong môi trường phân trang được thực hiện bởi các bit bảo vệ gán liền với mỗi khung. Thông thường, các bit này được giữ trong bảng trang. Một bit có thể định nghĩa một trang để đọc-viết hay chỉ đọc. Mỗi tham chiếu tới bộ nhớ sẽ tìm khắp bảng trang để xác định số khung tương ứng. Tại cùng thời điểm địa chỉ vật lý được tính , các bit bảo vệ có thể được kiểm tra để thẩm định rằng không có thao tác viết nào đang được thực hiện tới trang chỉ đọc. Cố gắng viết tới một trang chỉ đọc gây ra một trap phần cứng tới hệ điều hành (hay xung đột bộ nhớ bảo vệ).

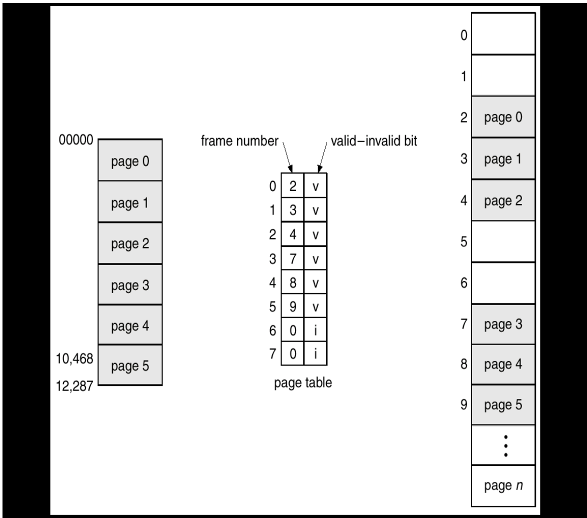
Chúng ta có thể dễ dàng mở rộng tiếp cận này để cung cấp một cấp độ bảo vệ chi tiết hơn. Chúng ta có thể tạo phần cứng để cung cấp bảo vệ chỉ đọc, đọc viết, chỉ thực thi. Hay bằng cách cung cấp các bit bảo vệ riêng cho mỗi loại truy xuất, chúng ta có thể cho phép bất cứ kết hợp của các truy xuất này; các cố gắng không hợp lệ sẽ được trap tới hệ điều hành.

Một bit nữa thường được gán tới mỗi mục từ trong bảng trang: một bit hợp lệ-không hợp lệ. Khi bit này được đặt là “hợp lệ” thì giá trị này hiển thị rằng trang được gán trong không gian địa chỉ luận lý bộ nhớ là trang hợp lệ. Nếu bit này được đặt là “không hợp lệ”, giá trị này hiển thị trang đó không ở trong không gian địa chỉ luận lý của quá trình. Các địa chỉ không hợp lệ được trap bằng cách sử dụng bit hợp lệ-không hợp lệ. Hệ điều hành thiết lập bit này cho mỗi trang để cho phép hay không cho phép truy xuất tới trang này. Thí dụ, trong một hệ thống với không gian địa chỉ 14 bit (0 tới 16383), chúng ta có thể có một chương trình sử dụng chỉ địa chỉ 0 tới 10468. Cho kích thước trang 2KB, chúng ta xem trường hợp trong hình VII-17. Địa chỉ trong các trang 0, 1, 2, 3, 4, và 5 thường được ánh xạ khắp bảng trang. Tuy nhiên, bất cứ những nỗ lực để tạo ra một địa chỉ trong trang 6 hay 7 nhận thấy rằng bit hợp lệ-không hợp lệ được đặt là không hợp lệ và máy tính sẽ trap tới hệ điều hành (tham chiếu trang không hợp lệ).

Vì chương trình mở rộng chỉ tới địa chỉ 10468, bất cứ tham chiếu vượt ra ngoài địa chỉ đó là không hợp lệ. Tuy nhiên, các tham chiếu tới trang 5 được xem là hợp lệ vì thế những địa chỉ tới 12287 là hợp lệ. Chỉ những địa chỉ từ 12288 tới 16383 là không hợp lệ. Vấn đề này là do kích thước trang 2KB và phản ánh phân mãnh trong của việc phân trang.

Rất hiếm khi một quá trình dùng tất cả dãy địa chỉ của nó. Thật vậy, nhiều quá trình dùng chỉ một phần nhỏ không gian địa chỉ còn trống cho chúng. Nó sẽ lãng phí rất nhiều trong những trường hợp này để tạo một bảng trang với các mục từ cho mỗi trang trong dãy địa chỉ. Hầu hết bảng này sẽ không được dùng nhưng sẽ mang đến không gian bộ nhớ có giá trị. Một số hệ thống cung cấp phần cứng, trong dạng một thanh ghi có chiều dài bảng trang (page-table length register-PTLR) để hiển thị kích thước của bảng trang. Giá trị này được kiểm tra

dựa trên mỗi địa chỉ luận lý để thẩm định địa chỉ đó nằm trong dãy địa chỉ hợp lệ cho quá trình. Lỗi của việc kiểm tra này gây ra một trap lỗi tới hệ điều hành.



Hình VII-16 Bit hợp lệ (v) và không hợp lệ (i) trong một bảng trang

Cấu trúc bảng trang

Trong phần này chúng ta sẽ xem xét một số kỹ thuật thông dụng nhất để xây dựng cấu trúc bảng trang.

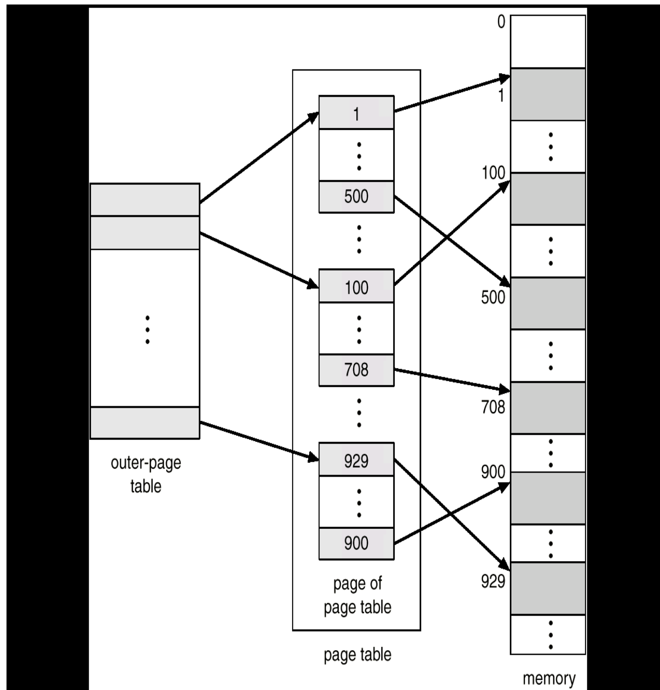
Bảng trang phân cấp

Hầu hết các hệ thống máy tính hiện đại hỗ trợ một không gian địa chỉ luận lý lớn (232 tới 264). Trong môi trường như thế, bảng trang trở nên quá lớn. Thí dụ, xét một hệ thống với không gian địa chỉ luận lý 32 bit. Nếu kích thước trang 4KB thì bảng trang có thể chứa tới 1 triệu mục từ (232/212). Giả sử rằng mỗi mục từ chứa 4 bytes, mỗi quá trình có thể cần tới 4MB không gian địa chỉ vật lý cho một bảng trang. Rõ ràng, chúng ta sẽ không muốn cấp phát bảng trang liên tiếp nhau. Một giải pháp đơn giản cho vấn đề này là chia bảng trang thành những phần nhỏ hơn. Có nhiều cách để đạt được sự phân chia này.

Một cách là dùng giải thuật phân trang hai cấp, trong đó bảng trang cũng được phân trang như hình VII-18.

Đây là thí dụ cho máy 32 bit với kích thước trang 4KB. Địa chỉ luận lý được chia thành số trang chứa 20 bit và độ dời trang chứa 12 bit. Vì chúng ta phân trang bảng trang, số trang được chia thành số trang 10 bit và độ dời trang 10-bit. Do đó, một địa chỉ luận lý như sau:

Số trang		Độ dời trang
P1	P2	d
10	10	12



Hình VII-17 Cơ chế bảng trang hai cấp

ở đây p1 là chỉ mục trong bảng trang bên ngoài và p2 là độ dời trong trang của bảng trang bên ngoài. Phương pháp dịch địa chỉ cho kiến trúc này được hiển thị trong hình VII-19. Vì dịch địa chỉ thực hiện từ những phần trong bảng trang bên ngoài, cơ chế này cũng được gọi là bảng trang được ánh xạ chuyển tiếp (forward-mapped page table). Petium-II sử dụng kiến trúc này.

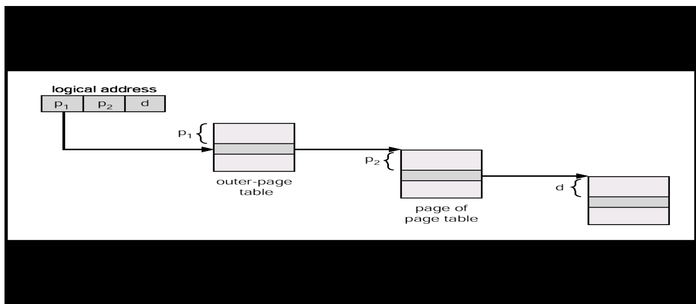
Kiến trúc VAX cũng hỗ trợ một biến dạng của phân trang hai cấp. VAX là máy 32-bit với kích thước trang 512 bytes. Không gian địa chỉ luận lý của một quá trình được chia làm 4 phần bằng nhau, mỗi phần chứa 230 bytes. Mỗi phần biểu diễn một phần khác nhau của không gian địa chỉ luận lý của một quá trình. Hai bit cao đầu tiên của địa chỉ luận lý chỉ rõ phần tương ứng. 21 bits tiếp theo biểu diễn số trang luận lý của phần đó, và 9 bits cuối biểu diễn độ dời trong trang mong muốn. Bằng cách chia bảng trang như thế, hệ điều hành có thể để những phân khu không được dùng cho tới khi một quá trình yêu cầu chúng. Một địa chỉ trên kiến trúc VAX như sau:

Phần	Trang	Độ dời
S	P	D
2	21	9

ở đây s chỉ rõ số phần, p là chỉ mục trong bảng trang và d là độ dời trong trang.

Kích thước của bảng trang cấp một cho một quá trình VAX dùng một phần vẫn là 221 bits * 4 bytes/trang = 8 MB. Để việc sử dụng bộ nhớ chính bị giảm nhiều hơn, VAX phân trang các bảng trang quá trình người dùng.

Đối với các hệ thống có không gian địa chỉ luận lý 64 bits, cơ chế phân trang hai cấp không còn phù hợp nữa. Để thể hiện điểm này, chúng ta giả sử rằng kích thước trang trong hệ thống là 4 KB (212). Trong trường hợp này, bảng trang sẽ chứa tới 252 mục từ. Nếu chúng ta dùng cơ chế phân trang hai cấp thì các bảng bên trong có thể là một trang dài chứa 210 mục từ. Các địa chỉ sẽ như thế này:



Hình VII-18 Dịch địa chỉ cho kiến trúc phân trang hai cấp 32-bit

Trang bên ngoài	Trang bên trong	Độ dời
P1	P2	D
42	10	12

Bảng trang bên ngoài sẽ chứa 242 mục từ, hay 244 bytes. Các phương pháp được chú trọng để tránh để trang lớn là chia bảng trang bên ngoài thành những phần nhỏ hơn. Tiếp cận này cũng được dùng trên một vài bộ xử lý 32-bit để thêm khả năng mềm dẽo và hiệu quả.

Chúng ta có thể chia bảng trang bên ngoài thành cơ chế phân trang 3 cấp. Giả sử rằng bảng trang bên ngoài được tạo ra từ các trang có kích thước chuẩn (210 mục từ, hay 212 bytes); một không gian địa chỉ 64 bit vẫn có kích thước rất lớn:

Trang bên ngoài cấp 2	Trang bên ngoài	Trang bên trong	Độ dời
P1	P2	P3	D
32	10	10	12

Bảng trang bên ngoài vẫn lớn 232.

Bước tiếp theo sẽ là cơ chế phân trang cấp bốn, ở đây bảng trang bên ngoài cấp hai cũng được phân trang. Kiến trúc SPARC (với 32-bit đánh địa chỉ) hỗ trợ cơ chế phân trang cấp ba, trái lại kiến trúc Motorola 68030 32-bit hỗ trợ cơ chế phân trang bốn cấp.

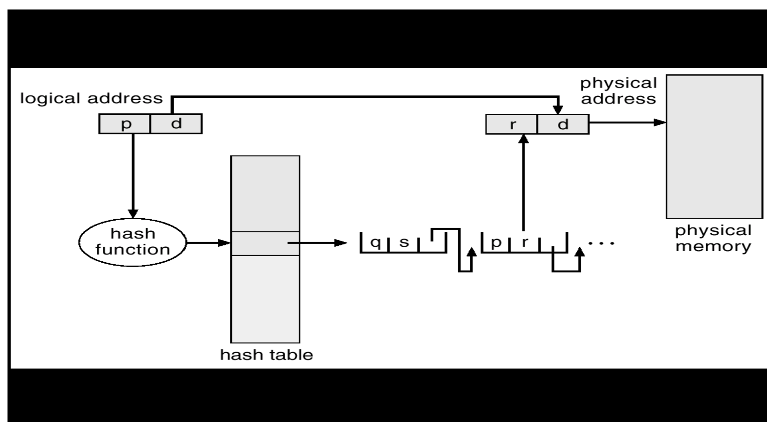
Tuy nhiên, đối với kiến trúc 64-bit, các bảng trang phân cấp thường được xem xét là không phù hợp. Thí dụ, UltraSPARC 64-bit sẽ yêu cầu phân trang bảy cấp – một số truy xuất bộ nhớ không được phép để dịch mỗi địa chỉ luận lý.

Bảng trang được băm

Một tiếp cận thông thường cho việc quản lý không gian địa chỉ lớn hơn 32-bit là dùng bảng trang được băm (hashed page table), với giá trị băm là số trang ảo. Mỗi mục từ trong bảng trang chứa một danh sách liên kết của các phần tử. Danh sách này băm tới cùng vị trí (để quản lý đụng độ). Mỗi phần tử chứa ba trường: (a) số trang ảo, (b) giá trị khung trang được ánh xạ và con trỏ chỉ tới phần tử kế tiếp trong danh sách liên kết.

Giải thuật thực hiện như sau: số trang ảo trong địa chỉ ảo được băm tới bảng băm. Số trang ảo được so sánh tới trường (a) trong phần tử đầu tiên của danh sách liên kết. Nếu có phần tử trùng khớp, khung trang tương ứng (trường (b) được dùng để hình thành địa chỉ vật lý mong muốn). Nếu không có phần tử nào trùng khớp, các mục từ tiếp theo trong danh sách liên kết được tìm kiếm số trang ảo trùng khớp. Cơ chế này được hiển thị trong hình VII-20 dưới đây:

Một biến thể đối với cơ chế này cho không gian địa chỉ 64-bit được đề nghị. Bảng trang được nhóm (Clustered page tables) tương tự như bảng băm ngoại trừ mỗi mục từ trong bảng băm tham chiếu tới nhiều trang (chẳng hạn như 16) hơn là một trang. Do đó, mục từ bảng trang đơn có thể lưu những ánh xạ cho nhiều khung trang vật lý. Bảng trang được nhóm đặc biệt có ích cho không gian địa chỉ rời nhau (spare), ở đó các tham chiếu bộ nhớ là không liên tục và tập hợp khắp không gian bộ nhớ.



Hình VII-19 Bảng trang được băm

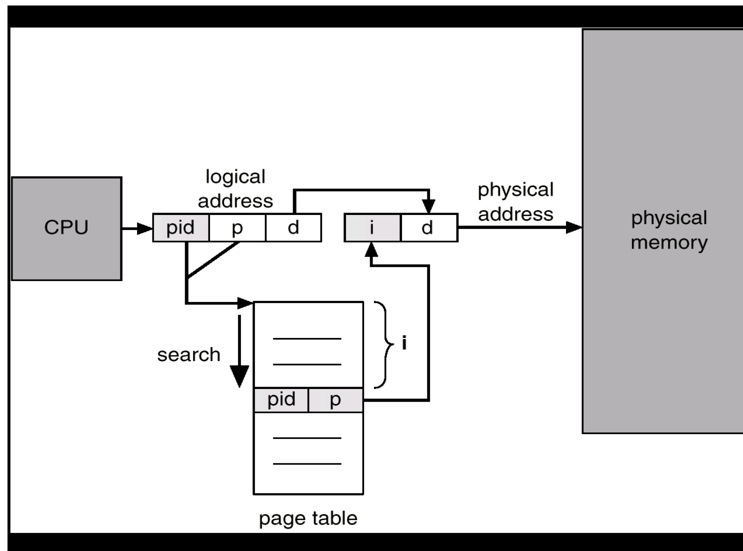
Bảng trang đảo

Thông thường, mỗi quá trình có một trang gán liền với nó. Bảng trang có một mục từ cho mỗi trang mà quá trình đó đang sử dụng (hay một khe cho mỗi địa chỉ ảo, không phụ thuộc tính hợp lệ sau đó). Biểu diễn bảng trang này là biểu diễn tự nhiên vì tham chiếu quá trình phân trang thông qua các địa chỉ ảo của trang. Sau đó, hệ điều hành phải dịch tham chiếu này vào một địa chỉ bộ nhớ vật lý. Vì bảng này được sắp xếp bởi địa chỉ ảo, hệ điều hành có thể tính toán nơi trong bảng mà mục từ địa chỉ vật lý được nối kết tới và sử dụng giá trị đó trực tiếp. Một trong những khó khăn của phương pháp này là mỗi bảng trang có thể chứa hàng triệu mục từ. Các bảng này có thể tiêu tốn lượng lớn bộ nhớ vật lý, được yêu cầu chỉ để giữ vết của bộ nhớ vật lý khác đang được sử dụng như thế nào.

Để giải quyết vấn đề này chúng ta có thể sử dụng một bảng trang đảo (inverted page table). Bảng trang đảo có một mục từ cho mỗi trang thật (hay khung) của bộ nhớ. Mỗi mục từ chứa địa chỉ ảo của trang được lưu trong

vị trí bộ nhớ thật đó, với thông tin về quá trình sở hữu trang đó. Do đó, chỉ một bảng trang trong hệ thống và nó chỉ có một mục từ cho mỗi trang của bộ nhớ vật lý. Hình VII-21 dưới đây hiển thị hoạt động của bảng trang đảo.

So sánh nó với hình VII-6, mô tả hoạt động của một bảng trang chuẩn. Vì chỉ một bảng trang trong hệ thống còn có nhiều không gian địa chỉ khác ánh xạ bộ nhớ vật lý, nên các bảng trang đảo thường yêu cầu một định danh không gian địa chỉ được lưu trong mỗi mục từ của bảng trang. Lưu trữ định danh không gian địa chỉ đảm bảo rằng ánh xạ của trang luận lý cho một quá trình xác định tới khung trang vật lý tương ứng. Thí dụ, hệ thống dùng bảng trang đảo gồm UltraSPARC 64-bit và PowerPC.



Hình VII-20 Bảng trang đảo

Để hiển thị phương pháp này, chúng ta mô tả một ấn bản được đơn giản hoá cài đặt bảng trang đảo dùng trong IBM RT. Mỗi địa chỉ ảo trong hệ thống chứa bộ ba:

<process-id, page-number, offset>.

Mỗi mục từ bảng trang đảo là một cặp <process-id, page-number>, ở đây process-id đảm bảo vai trò định danh không gian địa chỉ. Khi một tham chiếu bộ nhớ xảy ra, một phần của địa chỉ ảo, gồm <process-id, page-number>, được hiện diện trong hệ thống bộ nhớ. Sau đó, bảng trang đảo được tìm kiếm sự trùng khớp. Nếu sự trùng khớp được tìm thấy tại mục từ i thì địa chỉ vật lý <i, offset> được tạo ra. Nếu không tìm thấy thì một truy xuất địa chỉ không hợp lệ được cố gắng thực hiện.

Mặc dù cơ chế này giảm lượng bộ nhớ được yêu cầu để lưu mỗi bảng trang, nhưng nó gia tăng lượng thời gian cần cho việc tìm kiếm bảng khi có một tham chiếu xảy ra. Vì bảng trang đảo được lưu bởi địa chỉ vật lý nhưng tìm kiếm xảy ra trên địa chỉ ảo, toàn bộ bảng trang có thể cần được tìm kiếm sự trùng khớp. Sự tìm kiếm này có thể mất thời gian quá dài. Để làm giảm vấn đề này, chúng ta sử dụng một bảng băm được mô tả trong hình dưới đây để giới hạn việc tìm kiếm. Dĩ nhiên, mỗi truy xuất tới bảng băm thêm một tham chiếu tới thủ tục, để mà một tham chiếu bộ nhớ ảo yêu cầu ít nhất hai thao tác đọc bộ nhớ thật: một cho mục từ bảng băm và một cho bảng trang. Để cải tiến năng lực thực hiện, TLB được tìm kiếm đầu tiên, trước khi bảng băm được tra cứu.

Trang được chia sẻ

Một thuận lợi khác của phân trang là khả năng chia sẻ mã chung. Việc xem xét này đặc biệt quan trọng trong môi trường chia thời. Xét một hệ thống hỗ trợ 40 người dùng, mỗi người dùng thực thi một trình soạn thảo văn bản. Nếu trình soạn thảo văn bản chứa 150 KB mã và 50 KB dữ liệu, chúng ta sẽ cần 8000 KB để hỗ trợ 40 người dùng. Tuy nhiên, nếu mã là mã tái sử dụng (reentrant code), nó có thể được chia sẻ như được hiển thị trong hình VII-22. Ở đây chúng ta thấy một bộ soạn thảo ba trang-mỗi trang có kích thước 50 KB; kích thước trang lớn được dùng để đơn giản hoá hình này-đang được chia sẻ giữa ba quá trình. Mỗi quá trình có trang dữ liệu riêng của nó.

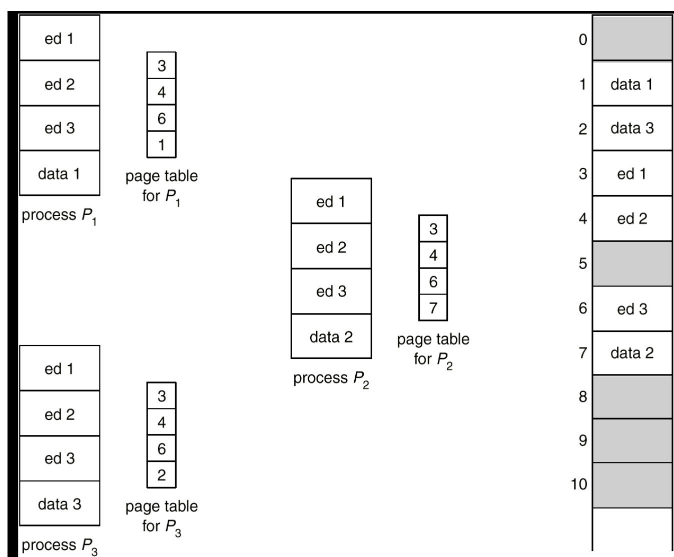
Mã tái sử dụng (hay thuần mã-pure code) là mã không thay đổi bởi chính nó. Nếu mã là tái sử dụng thì nó không bao giờ thay đổi trong quá trình thực thi. Do đó, hai hay nhiều quá trình có thể thực thi cùng mã tại cùng thời điểm. Mỗi quá trình có bản sao thanh ghi của chính nó và lưu trữ dữ liệu để quản lý dữ liệu cho việc thực thi của quá trình. Dĩ nhiên, dữ liệu cho hai quá trình khác nhau sẽ khác nhau cho mỗi quá trình.

Chỉ một bản sao của bộ soạn thảo cần được giữ trong bộ nhớ vật lý. Mỗi bảng trang của người dùng ánh xạ tới cùng bản sao vật lý của bộ soạn thảo nhưng các trang dữ liệu được ánh xạ tới các khung khác nhau. Do đó, để hỗ trợ 40 người dùng, chúng ta cần chỉ một bản sao của bộ soạn thảo (150 KB) cộng với 40 bản sao của 50 KB không gian dữ liệu trên một người dùng. Bây giờ toàn bộ không gian được yêu cầu là 2150 KB thay vì 8000 KB-một tiết kiệm lớn.

Những chương trình được dùng nhiều khác cũng có thể được chia sẻ - trình biên dịch, hệ thống cửa sổ, thư viện thời điểm thực thi, hệ cơ sở dữ liệu,…Để có thể chia sẻ, mã phải được tái sử dụng. Tính tự nhiên chỉ đọc của mã được chia sẻ sẽ không được phó mặc cho tính đúng đắn của mã; hệ điều hành nên tuân theo thuộc tính này. Chia sẻ bộ nhớ giữa các quá trình trên hệ điều hành tương tự chia sẻ không gian địa chỉ của một tác vụ bởi luồng. Ngoài ra, bộ nhớ được chia sẻ như một phương pháp giao tiếp liên quá trình. Một số hệ điều hành cài đặt bộ nhớ được chia sẻ dùng các trang được chia sẻ.

Hệ điều hành dùng bảng trang bên trong gặp khó khăn khi cài đặt bộ nhớ được chia sẻ. Bộ nhớ được chia sẻ thường được cài đặt như nhiều địa chỉ ảo (một địa chỉ cho mỗi quá trình chia sẻ bộ nhớ) mà chúng được ánh xạ tới một địa chỉ vật lý. Tuy nhiên, phương pháp chuẩn này không thể được dùng khi có chỉ một mục từ trang ảo cho mỗi trang vật lý vì thế một trang vật lý không thể có hai (hay nhiều) địa chỉ ảo được chia sẻ.

Tổ chức bộ nhớ dựa theo trang cung cấp nhiều lợi điểm khác để cho phép nhiều quá trình chia sẻ cùng trang vật lý.



Hình VII-21 chia sẻ mã trong môi trường phân trang

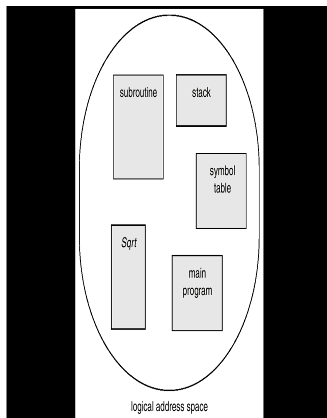
Phân đoạn

Một khía cạnh quan trọng của việc quản lý bộ nhớ mà trở nên không thể tránh với phân trang là ngăn cách tầm nhìn bộ nhớ của người dùng và bộ nhớ vật lý thật sự. Tầm nhìn bộ nhớ của người dùng không giống như bộ nhớ vật lý. Tầm nhìn người dùng được ánh xạ vào bộ nhớ vật lý. Việc ánh xạ cho phép sự khác nhau giữa bộ nhớ luận lý và bộ nhớ vật lý.

Phương pháp cơ bản

Người dùng nghĩ bộ nhớ như mảng tuyến tính các byte, một số byte chứa chỉ thị lệnh, một số khác chứa dữ liệu hay không? Hầu hết mọi người nói không. Đúng hơn là, người dùng thích nhìn bộ nhớ như tập hợp các phân đoạn có kích thước thay đổi, và không cần xếp thứ tự giữa các phân đoạn (như hình VII-23).

Chúng ta nghĩ như thế nào về một chương trình khi chúng ta đang viết nó? Chúng ta nghĩ nó như một chương trình chính với một tập hợp các chương trình con, thủ tục, hàm, hay các module. Có thể có các cấu trúc dữ liệu khác nhau: bảng, mảng, ngăn xếp, biến,..Mỗi module hay thành phần dữ liệu này được tham chiếu bởi tên. Chúng ta nói "bảng danh biểu", "hàm sqrt", "chương trình chính" không quan tâm đến địa chỉ trong bộ nhớ mà những phần tử này chiếm. Chúng ta không quan tâm bảng danh biểu được lưu trữ trước hay sau hàm sqrt. Mỗi phân đoạn này có chiều dài thay đổi; thực chất chiều dài được định nghĩa bởi mục đích của phân đoạn trong chương trình. Các phần tử trong một phân đoạn được định nghĩa bởi độ dời của chúng từ điểm bắt đầu của phân đoạn: lệnh đầu tiên của chương trình, mục từ thứ mười bảy trong bảng danh biểu, chỉ thị thứ năm của hàm sqrt,…



Hình VII-22 Tầm nhìn chương trình của người dùng

Phân đoạn là một cơ chế quản lý bộ nhớ hỗ trợ tầm nhìn bộ nhớ của người dùng. Không gian địa chỉ luận lý là tập hợp các phân đoạn. Mỗi phân đoạn có tên và chiều dài. Các địa chỉ xác định tên phân đoạn và độ dời trong phân đoạn. Do đó, người dùng xác định mỗi địa chỉ bằng hai lượng: tên phân đoạn và độ dời. (tương phản cơ chế này với cơ chế phân trang, trong đó người dùng chỉ xác định một địa chỉ đơn, được chia bởi phần cứng thành số trang và độ dời, tất cả không thể nhìn thấy đối với người lập trình).

Để đơn giản việc cài đặt, các phân đoạn được đánh số và được tham chiếu tới bởi số phân đoạn, hơn là bởi tên phân đoạn. Do đó, địa chỉ luận lý chứa một bộ hai:

<số phân đoạn, độ dời>

Thông thường, chương trình người dùng được biên dịch, và trình biên dịch tự động tạo ra các phân đoạn phản ánh chương trình nhập. Một chương trình Pascal có thể tạo các phân đoạn riêng như sau:

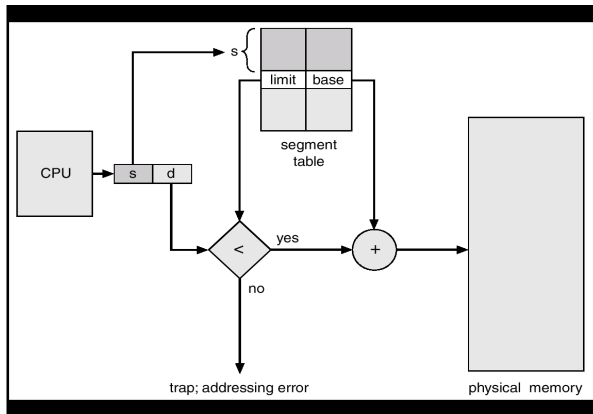
1. Các biến toàn cục;
2. Ngăn xếp gọi thủ tục, để lưu trữ các tham số và trả về các địa chỉ;
3. Phần mã của mỗi thủ tục hay hàm;
4. Các biến cục bộ của mỗi thủ tục và hàm

Một trình biên dịch có thể tạo một phân đoạn riêng cho mỗi khối chung. Các mảng có thể được gán các phân đoạn riêng. Bộ nạp có thể mang tất cả phân đoạn này và gán chúng số phân đoạn.

Phần cứng

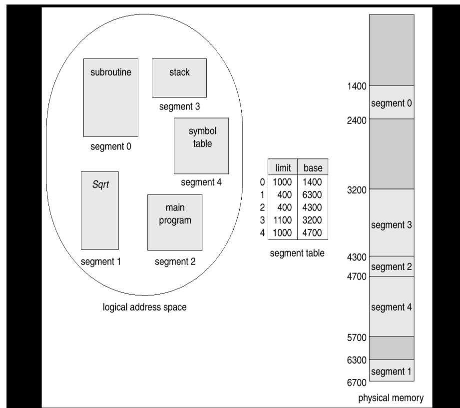
Mặc dù người dùng có thể tham chiếu tới các đối tượng trong chương trình bởi một địa chỉ hai chiều, bộ nhớ vật lý là chuỗi một chiều các byte. Do đó, chúng ta phải xác định việc cài đặt để ánh xạ địa chỉ hai chiều được định nghĩa bởi người dùng vào địa chỉ vật lý một chiều. Ánh xạ này được tác động bởi một bảng phân đoạn. Mỗi mục từ của bảng phân đoạn có một nền phân đoạn (segment base) và giới hạn phân đoạn (segment limit). Nền phân đoạn chứa địa chỉ vật lý bắt đầu, nơi phân đoạn định vị trong bộ nhớ, ngược lại giới hạn phân đoạn xác định chiều dài của phân đoạn.

Sử dụng bảng phân đoạn được hiển thị như hình VII-24. Một địa chỉ luận lý có hai phần: số phân đoạn s và độ dời phân đoạn d. Số phân đoạn được dùng như chỉ mục trong bảng đoạn. Độ dời d của địa chỉ luận lý phải ở trong khoảng từ 0 tới giới hạn đoạn. Nếu không chúng ta sẽ trap tới hệ điều hành (địa chỉ vật lý vượt qua điểm cuối của phân đoạn). Nếu độ dời này là hợp lệ thì nó được cộng thêm giá trị nền của phân đoạn để tạo ra địa chỉ trong bộ nhớ vật lý của byte mong muốn. Do đó, bảng phân đoạn là một mảng của cặp thanh ghi nền và giới hạn.



Hình VII-23 Phần cứng phân đoạn

Xét trường hợp như hình VII-25. Chúng ta có năm phân đoạn được đánh số từ 0 đến 4. Các phân đoạn được lưu trong bộ nhớ vật lý như được hiển thị. Bảng phân đoạn có một mục từ riêng cho mỗi phân đoạn, cho địa chỉ bắt đầu của phân đoạn trong bộ nhớ vật lý (hay nền) và chiều dài của phân đoạn đó (hay giới hạn). Thí dụ, phân đoạn 2 dài 400 bytes và bắt đầu tại vị trí 4300. Do đó, một tham chiếu byte 53 của phân đoạn 2 được ánh xạ tới vị trí 4300 + 53 = 4353. Một tham chiếu tới phân đoạn 3, byte 852, được ánh xạ tới 3200 (giá trị nền của phân đoạn 3) +852=4052. Một tham chiếu tới byte 1222 của phân đoạn 0 dẫn đến một trap tới hệ điều hành, khi phân đoạn này chỉ dài 1000 bytes.

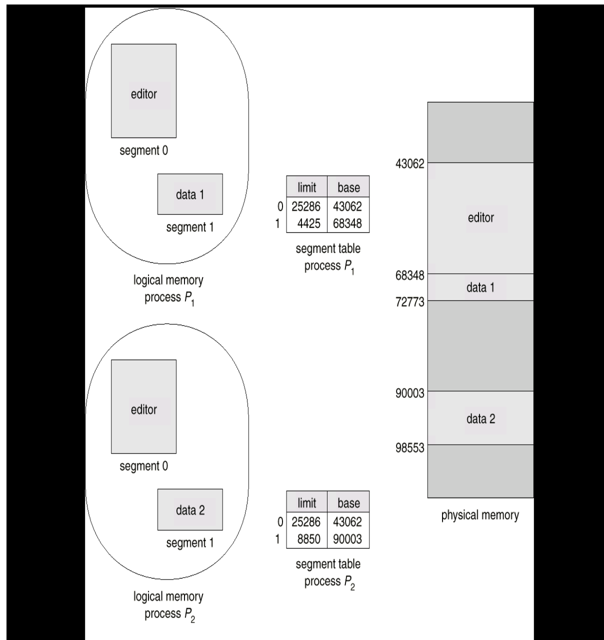


Hình VII-24 Thí dụ về phân đoạn

Bảo vệ và chia sẻ

Lợi điểm đặc biệt của phân đoạn là sự gắn liền bảo vệ với các phân đoạn. Vì các phân đoạn biểu diễn một phần được định nghĩa của chương trình, tương tự như tất cả mục từ trong phân đoạn sẽ được dùng cùng một cách. Do đó, một số phân đoạn là chỉ thị, trong khi một số phân đoạn khác là dữ liệu. Trong một kiến trúc hiện đại, các chỉ thị không hiệu chỉnh chính nó vì thế các phân đoạn chỉ thị có thể được định nghĩa như chỉ đọc hay chỉ thực thi. Phần cứng ánh xạ bộ nhớ sẽ kiểm tra các bits bảo vệ được gắn với mỗi mục từ trong bảng phân đoạn để ngăn chặn các truy xuất không hợp lệ tới bộ nhớ, như cố gắng viết tới phân đoạn chỉ đọc hay sử dụng những phân đoạn chỉ đọc như dữ liệu. Bằng cách thay thế một mảng trong phân đoạn của chính nó, phần cứng quản lý bộ nhớ sẽ tự động kiểm tra các chỉ số của mảng là hợp lệ và không vượt ra ngoài giới hạn của mảng. Do đó, nhiều lỗi chương trình sẽ được phát hiện bởi phần cứng trước khi chúng có thể gây ra tác hại lớn.

Một lợi điểm khác liên quan đến chia sẻ mã hay dữ liệu. Mỗi quá trình có một bảng phân đoạn gắn với nó. Bộ phân phát dùng bảng phân đoạn này để định nghĩa phân đoạn phần cứng khi một quá trình được cấp CPU. Các phân đoạn được chia sẻ khi các mục từ trong bảng phân đoạn của hai quá trình khác nhau chỉ tới cùng một vị trí vật lý (như hình VII-26).



Hình VII-25 Chia sẻ các phân đoạn trong một hệ thống bộ nhớ được phân đoạn

Chia sẻ xảy ra tại cấp phân đoạn. Do đó, bất cứ thông tin có thể được chia sẻ nếu nó được định nghĩa là một phân đoạn. Một số phân đoạn có thể được chia sẻ vì thế một chương trình được hình thành từ nhiều phân đoạn có thể được chia sẻ.

Thí dụ, xét việc sử dụng một trình soạn thảo văn bản trong hệ thống chia thời. Trình soạn thảo hoàn chỉnh có thể rất lớn, được hình thành từ nhiều phân đoạn có thể được chia sẻ giữa tất cả người dùng, giới hạn địa chỉ vật lý được yêu cầu hỗ trợ các tác vụ soạn thảo. Thay vì n bản sao của trình soạn thảo, chúng ta chỉ cần một bản sao. Đối với mỗi người dùng, chúng ta vẫn cần các phân đoạn riêng, duy nhất để lưu các biến cục bộ. Dĩ nhiên, các phân đoạn này sẽ không được chia sẻ.

Chúng ta cũng có thể chia sẻ một số phần chương trình. Thí dụ, các gói chương trình con dùng chung có thể được chia sẻ giữa nhiều người dùng nếu chúng được định nghĩa như các phân đoạn chia sẻ, chỉ đọc. Thí dụ, hai chương trình Fortran có thể dùng cùng hàm Sqrt, nhưng chỉ một bản sao vật lý của hàm Sqrt được yêu cầu.

Mặc dù việc chia sẻ này có vẻ đơn giản, nhưng có những xem xét tinh tế. Điển hình, phân đoạn mã chứa các tham chiếu tới chính nó. Thí dụ, một lệnh nhảy (jump) có điều kiện thường có một địa chỉ chuyển gồm số phân đoạn và độ dời. Số phân đoạn của địa chỉ chuyển sẽ là số phân đoạn của phân đoạn mã. Nếu chúng ta cố gắng chia sẻ phân đoạn này, tất cả quá trình chia sẻ phải định nghĩa phân đoạn mã được chia sẻ để có cùng số phân đoạn.

Thí dụ, nếu chúng ta muốn chia sẻ hàm Sqrt và một quá trình muốn thực hiện nó phân đoạn 4 và một quá trình khác muốn thực hiện nó phân đoạn 17, hàm Sqrt nên tham chiếu tới chính nó như thế nào? Vì chỉ có một bản sao vật lý của Sqrt, nó phải được tham chiếu tới chính nó trong cùng cách cho cả hai người dùng-nó phải có một số phân đoạn duy nhất. Khi số người dùng chia sẻ tăng do đó khó khăn trong việc tìm số phân đoạn có thể chấp nhận cũng tăng.

Các phân đoạn chỉ đọc không chứa con trỏ vật lý có thể được chia sẻ như số phân đoạn khác nhau, như các phân đoạn mã tham chiếu chính nó không trực tiếp. Thí dụ, các nhánh điều kiện xác định địa chỉ nhánh như một độ dời từ bộ đếm chương trình hiện hành hay quan hệ tới thanh ghi chứa số phân đoạn hiện hành nên cho phép mã tránh tham chiếu trực tiếp tới số phân đoạn hiện hành.

Sự phân mãnh

Bộ định thời biểu dài phải tìm và cấp phát bộ nhớ cho tất cả các phân đoạn của chương trình người dùng. Trường hợp này tương tự như phân trang ngoại trừ các phân đoạn có chiều dài thay đổi; các trang có cùng kích thước. Do đó, với cơ chế phân khu có kích thước thay đổi, cấp phát bộ nhớ là một vấn đề cấp phát lưu trữ động, thường giải quyết với giải thuật best-fit hay first-fit.

Phân đoạn có thể gây ra sự phân mãnh, khi tất cả khối bộ nhớ trống là quá nhỏ để chứa một phân đoạn. Trong trường hợp này, quá trình có thể phải chờ cho đến khi nhiều bộ nhớ hơn (hay ít nhất một lỗ lớn hơn) trở nên sẵn dùng, hay cho tới khi việc hợp nhất các lỗ nhỏ để tạo một lỗ lớn hơn. Vì sự phân đoạn dùng giải thuật tái định vị động nên chúng ta có thể gom bộ nhớ bất cứ khi nào chúng ta muốn. Nếu bộ định thời biểu CPU phải chờ một quá trình vì vấn đề cấp phát bộ nhớ, nó có thể (hay không thể) bỏ qua hàng đợt CPU để tìm một quá trình nhỏ hơn, có độ ưu tiên thấp hơn để chạy.

Phân mãnh ngoài đối với cơ chế phân đoạn là vấn đề quan trọng như thế nào? Định thời biểu theo thuật ngữ dài với sự cô đặc sẽ giúp giải quyết vấn đề phân mãnh phải không? Câu trả lời phụ thuộc vào kích thước trung bình của phân đoạn. Ở mức độ cao nhất, chúng ta có thể định nghĩa mỗi quá trình là một phân đoạn. Tiếp cận này cắt giảm cơ chế phân khu có kích thước thay đổi. Ở cấp độ khác, mỗi byte có thể được đặt trong chính phân đoạn của nó và được cấp phát riêng. Sắp xếp này xoá đi việc phân mãnh bên ngoài; tuy nhiên mỗi byte cần một thanh ghi nền cho mỗi tái định vị của nó, gấp đôi bộ nhớ được dùng! Dĩ nhiên, bước luận lý tiếp theo-các phân đoạn nhỏ có kích thước cố định-là phân trang. Thông thường, nếu kích thước phân đoạn trung bình là nhỏ, phân mãnh ngoài cũng sẽ nhỏ. Vì cá nhân các phân đoạn là nhỏ hơn toàn bộ quá trình nên chúng có vẻ thích hợp hơn để đặt vào trong các khối bộ nhớ sẵn dùng.

Phân đoạn với phân trang

Cả hai phân đoạn và phân trang có những lợi điểm và nhược điểm. Thật vậy, hai bộ vi xử lý phổ biến nhất hiện nay là: dòng Motorola 68000 được thiết kế dựa trên cơ sở không gian địa chỉ phẳng, ngược lại, họ Intel 80x86 và Petium dựa trên cơ sở phân đoạn. Cả hai là mô hình bộ nhớ hợp nhất hướng tới sự kết hợp của phân trang và phân đoạn. Chúng ta có thể kết hợp hai phương pháp để tận dụng lợi điểm của chúng. Sự kết hợp này được thể hiện tốt nhất bởi kết trúc của Intel 386.

Ấn bản IBM OS/2 32-bit là một hệ điều hành chạy trên đỉnh của kiến trúc Intel 386 (hay cao hơn). Intel 386 sử dụng phân đoạn với phân trang cho việc quản lý bộ nhớ. Số tối đa các phân đoạn trên quá trình là 16KB và mỗi phân đoạn có thể lớn tới 4GB. Kích thước trang là 4 KB. Chúng ta sẽ không cho một mô tả đầy đủ về cấu trúc quản lý bộ nhớ của Intel 386 trong giáo trình này. Thay vào đó, chúng ta sẽ trình bày các ý tưởng quan trọng.

Không gian địa chỉ luận lý của quá trình được chia thành hai phân khu. Phân khu thứ nhất chứa tới 8 KB các phân đoạn dành riêng cho quá trình đó. Phân khu thứ hai chứa tới 8 KB các phân đoạn được chia sẻ giữa tất cả quá trình. Thông tin về phân khu thứ nhất được giữ trong bảng mô tả cục bộ (Local Descriptor Table-LDT), thông tin về phân khu thứ hai được giữ trong bảng mô tả toàn cục (Global Descriptor Table-GDL). Mỗi mục từ trong LDT và GDT chứa 8 bytes, với thông tin chi tiết về phân đoạn xác định gồm vị trí nền và chiều dài của phân đoạn đó.

Địa chỉ luận lý là một cặp (bộ chọn, độ dời), ở đây bộ chọn là một số 16-bit

S	G	P
13	1	2

Trong đó: s gán tới số phân đoạn, g hiển thị phân đoạn ở trong GDT hay LDT, và p giải quyết vấn đề bảo vệ. Độ dời là một số 32-bit xác định vị trí của byte (hay từ) trong phân đoạn.

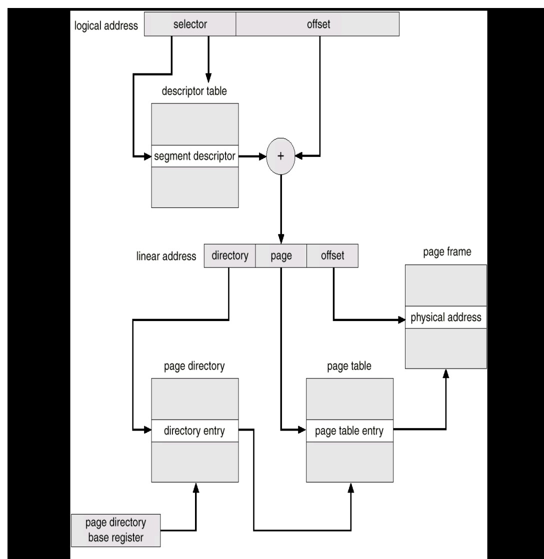
Máy này có 6 thanh ghi, cho phép 6 phân đoạn được xác định tại bất cứ thời điểm nào bởi một quá trình. Nó có 6 thanh ghi vi chương trình 8-byte để quản lý bộ mô tả tương ứng từ LDT hay GDT. Bộ lưu trữ này để Intel 386 tránh phải đọc bộ mô tả từ bộ nhớ cho mỗi lần tham chiếu bộ nhớ.

Địa chỉ vật lý trên 386 dài 32 bits và được hình thành như sau. Thanh ghi đoạn chỉ tới mục từ tương ứng trong LDT hay GDT. Thông tin nền và giới hạn về phân đoạn được dùng để phát sinh một địa chỉ tuyến tính. Đầu tiên, giới hạn được dùng để kiểm tra tính hợp lệ của địa chỉ. Nếu địa chỉ không hợp lệ, lỗi bộ nhớ được tạo ra, dẫn đến một trap tới hệ điều hành. Nếu nó là hợp lệ, thì giá trị của độ dời được cộng vào giá trị của nền, dẫn đến địa chỉ tuyến tính 32-bit. Sau đó, địa chỉ này được dịch thành địa chỉ vật lý.

Như được nêu trước đó, mỗi phân đoạn được phân trang và mỗi trang có kích thước 4 KB. Do đó, bảng trang có thể chứa tới 1 triệu mục từ. Vì mỗi mục từ chứa 4 bytes nên mỗi quá trình có thể cần 4 MB không gian địa chỉ vật lý cho một bảng trang. Rõ ràng, chúng ta không muốn cấp phát bảng trang liên tục trong bộ nhớ. Giải pháp này được thông qua trong Intel 386 để dùng cơ chế phân trang 2 cấp. Địa chỉ tuyến tính được chia thành số trang chứa 20 bits, và độ dời trang chứa 12 bits. Vì chúng ta phân trang bảng trang, số trang được chia nhỏ thành con trỏ thư mục trang 10-bit và con trỏ bảng trang 10-bit. Địa chỉ luận lý như sau:

P1	P2	D
10	10	12

Cơ chế dịch địa chỉ cho kiến trúc này tương tự như cơ chế được hiển thị trong hình VII-18. Dịch địa chỉ Intel được hiển thị chi tiết hơn trong hình VII-27dưới đây.



Hình VII-26 Dịch địa chỉ Intel 386

Để cải tiến tính hiệu quả của việc sử dụng bộ nhớ vật lý, bảng trang Intel 386 có thể được hoán vị tới đĩa. Trong trường hợp này, mỗi bit được dùng trong mục từ thư mục trang để hiển thị bảng mà mục từ đang chỉ tới ở trong bộ nhớ hay trên đĩa. Nếu bảng ở trên đĩa, hệ điều hành có thể dùng 31 bit còn lại để xác định vị trí đĩa của bảng; sau đó bảng có thể được mang vào bộ nhớ theo yêu cầu.

Tóm tắt

Các giải thuật quản lý bộ nhớ cho hệ điều hành đa chương trải dài từ tiếp cận hệ thống người dùng đơn tới phân đoạn được phân trang. Yếu tố quyết định lớn nhất của phương pháp được dùng trong hệ thống cụ thể là phần cứng được cải thiện. Mỗi địa chỉ bộ nhớ được được tạo ra bởi CPU phải được kiểm tra hợp lệ và có thể được ánh xạ tới một địa chỉ vật lý. Kiểm tra không thể được cài đặt (hữu hiệu) bằng phần mềm. Do đó, chúng ta bị ràng buộc bởi tính sẵn dùng phần cứng.

Các giải thuât quản lý bộ nhớ được thảo luận (cấp phát liên tục, phân trang, phân đoạn, và sự kết hợp của phân trang và phân đoạn) khác nhau trong nhiều khía cạnh. Trong so sánh các chiến lược quản lý bộ nhớ, chúng ta nên sử dụng các xem xét sau:

- Hỗ trợ phần cứng: thanh ghi nền hay cặp thanh ghi nền và thanh ghi giới hạn là đủ cho cơ chế phân khu đơn và đa, ngược lại phân trang và phân đoạn cần bảng ánh xạ để xác định ánh xạ địa chỉ.
- Năng lực: khi giải thuật quản lý bộ nhớ trở nên phức tạp hơn thì thời gian được yêu cầu để ánh xạ địa chỉ luận lý tới địa chỉ vật lý tăng. Đối với các hệ thống đơn giản, chúng ta chỉ cần so sánh hay cộng địa chỉ luận lý-các thao tác này phải nhanh. Phân trang và phân đoạn có thể nhanh như nếu bảng được cài đặt trong các thanh ghi nhanh. Tuy nhiên, nếu bảng ở trong bộ nhớ thì về thực chất truy xuất bộ nhớ của người dùng có thể bị giảm. Một TLB có thể hạn chế việc giảm năng lực tới mức có thể chấp nhận.
- Phân mãnh: một hệ thống đa chương sẽ thực hiện hiệu quả hơn nếu nó có cấp độ đa chương cao hơn. Đối với một tập hợp quá trình được cho, chúng ta có thể tăng cấp độ đa chương chỉ bằng cách đóng gói nhiều quá trình hơn trong bộ nhớ. Để hoàn thành tác vụ này, chúng ta phải giảm sự lãng phí hay phân mãnh bộ nhớ. Các hệ thống với các đơn vị cấp phát có kích thước cố định, như cơ chế đơn phân khu và phân trang, gặp phải sự phân mãnh trong. Các hệ thống với đơn vị cấp phát có kích thước thay đổi như cơ chế đa phân khu và phân đoạn, gặp phải sự phân mãnh ngoài.
- Tái định vị: một giải pháp cho vấn đề phân mãnh ngoài là cô đặc. Cô đặc liên quan đến việc chuyển dịch một chương trình trong bộ nhớ không chú ý những thay đổi của chương trình. Xem xét này yêu cầu địa chỉ luận lý được tái định vị động tại thời điểm thực thi. Nếu địa chỉ được tái định vị chỉ tại thời điểm nạp, chúng ta không thể lưu trữ dạng cô đặc.
- Hoán vị: bất cứ giải thuật có thể có hoán vị được thêm tới nó. Tại những khoảng thời gian được xác định bởi hệ điều hành, thường được mô tả bởi các chính xác định thời, các quá trình được chép từ bộ nhớ chính tới vùng lưu trữ phụ và sau đó được chép trở lại tới bộ nhớ chính. Cơ chế này cho phép nhiều quá trình được chạy hơn là có thể đặt vào bộ nhớ tại cùng một thời điểm.
- Chia sẻ: một phương tiện khác để gia tăng cấp độ đa chương là chia sẻ mã và dữ liệu giữa các người dùng khác nhau. Thường việc chia sẻ yêu cầu phân trang hay phân đoạn được dùng, để cung cấp những gói thông tin nhỏ (các trang hay các đoạn) có thể được chia sẻ. Chia sẻ là một phương tiện chạy nhiều quá trình với lượng bộ nhớ giới hạn nhưng các chương trình và dữ liệu được chia sẻ phải được thiết kế cẩn thận.
- Bảo vệ: nếu phân trang hay phân đoạn được cung cấp, các phần khác nhau của chương trình người dùng có thể được khai báo chỉ thực thi, chỉ đọc, hay đọc-viết. Sự hạn chế này là cần thiết với mã và dữ liệu được chia sẻ và thường có ích trong bất cứ trường hợp nào để cung cấp việc kiểm tra tại thời gian thực thi cho các lỗi lập trình thông thường.

Bộ nhớ ảo
1 Mục đích Sau khi học xong chương này, người học nắm được những kiến thức sau: - Hiểu kỹ thuật bộ nhớ ảo - Hiểu bộ nhớ ảo ở dạng phân trang theo yêu cầu - Hiểu độ phức tạp và chi phí trong từng kỹ thuật để cài đặt bộ nhớ ảo

BỘ NHỚ ẢO

Mục đích

Sau khi học xong chương này, người học nắm được những kiến thức sau:

- Hiểu kỹ thuật bộ nhớ ảo
- Hiểu bộ nhớ ảo ở dạng phân trang theo yêu cầu
- Hiểu độ phức tạp và chi phí trong từng kỹ thuật để cài đặt bộ nhớ ảo

Giới thiệu

Trong chương trước, chúng ta thảo luận các chiến lược quản lý bộ nhớ được dùng trong hệ thống máy tính. Tất cả những chiến lược này có cùng mục đích: giữ nhiều quá trình trong bộ nhớ cùng một lúc để cho phép đa chương. Tuy nhiên, chúng có khuynh hướng yêu cầu toàn bộ quá trình ở trong bộ nhớ trước khi quá trình có thể thực thi.

Bộ nhớ ảo là một kỹ thuật cho phép việc thực thi của quá trình mà quá trình có thể không hoàn toàn ở trong bộ nhớ. Một lợi điểm quan trọng của cơ chế này là các chương trình có thể lớn hơn bộ nhớ vật lý. Ngoài ra, bộ nhớ ảo phóng đại bộ nhớ chính thành bộ nhớ luận lý cực lớn khi được hiển thị bởi người dùng. Kỹ thuật này giải phóng người lập trình từ việc quan tâm đến giới hạn kích thước bộ nhớ. Bộ nhớ ảo cũng cho phép các quá trình dễ dàng chia sẻ tập tin và không gian địa chỉ, cung cấp cơ chế hữu hiện cho việc tạo quá trình.

Tuy nhiên, bộ nhớ ảo không dễ cài đặt và về thực chất có thể giảm năng lực nếu nó được dùng thiếu thận trọng. Trong chương này, chúng ta thảo luận bộ nhớ ảo trong dạng phân trang theo yêu cầu và xem xét độ phức tạp và chi phí.

Kiến thức nền

Các giải thuật quản lý bộ nhớ trong chương trước là cần thiết vì một yêu cầu cơ bản: các chỉ thị đang được thực thi phải ở trong bộ nhớ vật lý. Tiếp cận đầu tiên để thoả mãn yêu cầu này đặt toàn bộ không gian địa chỉ luận lý trong bộ nhớ vật lý. Phủ lắp và nạp động có thể giúp làm giảm hạn chế này nhưng chúng thường yêu cầu sự đề phòng đặc biệt và công việc phụ thêm bởi người lập trình. Hạn chế này dường như cần thiết và phù hợp nhưng nó không may mắn vì nó giới hạn kích thước của một chương trình đối với kích thước bộ nhớ vật lý.

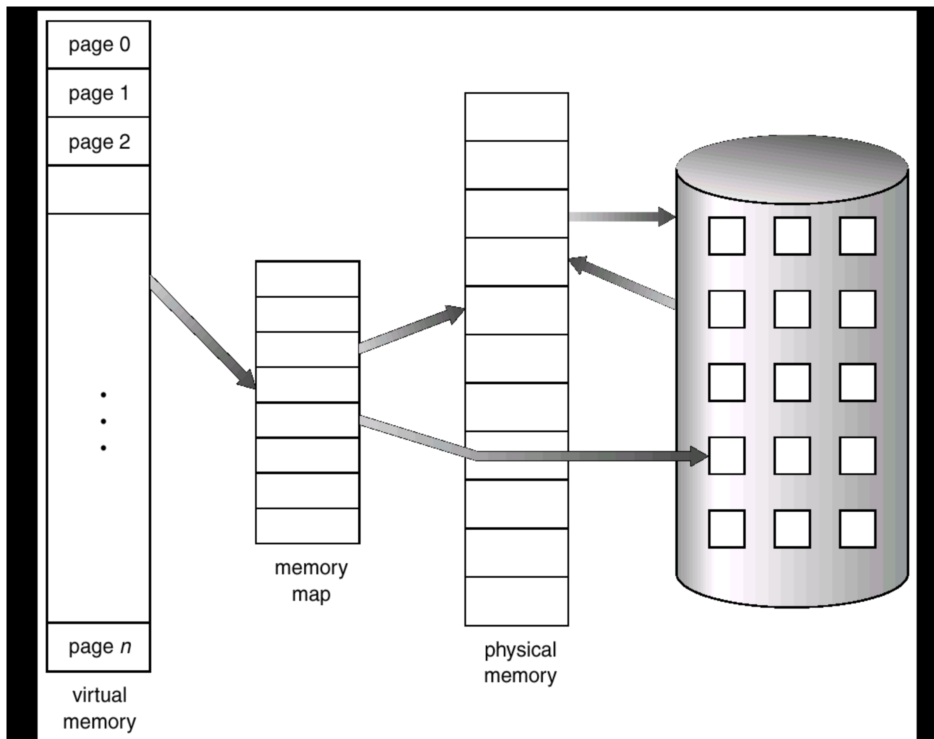
Thật vậy, xem xét các chương trình thực thi chúng ta nhận thấy rằng trong nhiều trường hợp toàn bộ chương trình là không cần thiết. Thậm chí trong những trường hợp toàn bộ chương trình được yêu cầu nhưng không phải tất cả chương trình được yêu cầu cùng một lúc.

Khả năng thực thi chương trình chỉ một phần chương trình ở trong bộ nhớ có nhiều lợi điểm:

- Chương trình sẽ không còn bị ràng buộc bởi không gian bộ nhớ vật lý sẵn có. Người dùng có thể viết chương trình có không gian địa chỉ ảo rất lớn, đơn giản hoá tác vụ lập trình.
- Vì mỗi chương trình người dùng có thể lấy ít hơn bộ nhớ vật lý nên nhiều chương trình hơn có thể được thực thi tại một thời điểm. Điều này giúp gia tăng việc sử dụng CPU và thông lượng nhưng không tăng thời gian đáp ứng.
- Yêu cầu ít nhập/xuất hơn để nạp hay hoán vị mỗi chương trình người dùng trong bộ nhớ vì thế mỗi chương trình người dùng sẽ chạy nhanh hơn.

Do đó, chạy một chương trình mà nó không nằm hoàn toàn trong bộ nhớ có lợi cho cả người dùng và hệ thống.

Bộ nhớ ảo là sự tách biệt bộ nhớ luận lý từ bộ nhớ vật lý. Việc tách biệt này cho phép bộ nhớ ảo rất lớn được cung cấp cho người lập trình khi chỉ bộ nhớ vật lý nhỏ hơn là sẵn dùng (hình VIII-1). Bộ nhớ ảo thực hiện tác vụ lập trình dễ hơn nhiều vì người lập trình không cần lo lắng về lượng bộ nhớ vật lý sẵn có nữa hay về mã gì có thể được thay thế trong việc phủ lắp; thay vào đó, người lập trình có thể quan tâm vấn đề được lập trình. Trên những hệ thống hỗ trợ bộ nhớ ảo, việc phủ lắp hầu như biến mất.



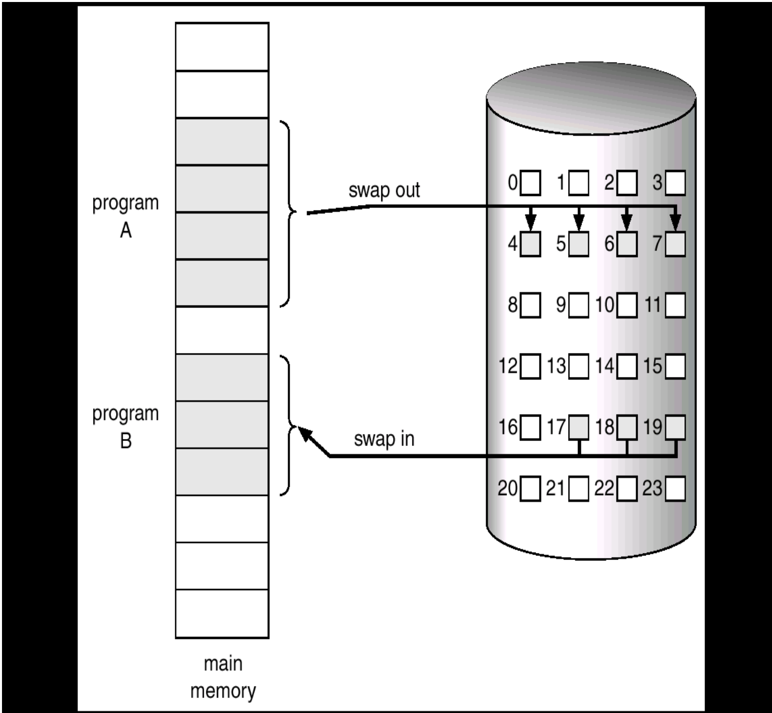
Hình VIII-1 Lưu đồ minh hoạ bộ nhớ ảo lơn hơn bộ nhớ vật lý

Thêm vào đó, việc tách biệt bộ nhớ luận lý từ bộ nhớ vật lý, bộ nhớ ảo cũng cho phép các tập tin và bộ nhớ được chia sẻ bởi những quá trình khác nhau thông qua việc chia sẻ trang. Ngoài ra, chia sẻ trang cho phép cải tiến năng lực trong khi tạo quá trình.

Bộ nhớ ảo thường được cài đặt bởi phân trang theo yêu cầu (demand paging). Nó cũng có thể được cài đặt trong cơ chế phân đoạn. Một vài hệ thống cung cấp cơ chế phân đoạn được phân trang. Trong cơ chế này các phân đoạn được chia thành các trang. Do đó, tầm nhìn người dùng là phân đoạn, nhưng hệ điều hành có thể cài đặt tầm nhìn này với cơ chế phân trang theo yêu cầu. Phân đoạn theo yêu cầu cũng có thể được dùng để cung cấp bộ nhớ ảo. Các hệ thống máy tính của Burrough dùng phân đoạn theo yêu cầu. Tuy nhiên, các giải thuật thay thế đoạn phức tạp hơn các giải thuật thay thế trang vì các đoạn có kích thước thay đổi. Chúng ta không đề cập phân đoạn theo yêu cầu trong giáo trình này.

Phân trang theo yêu cầu

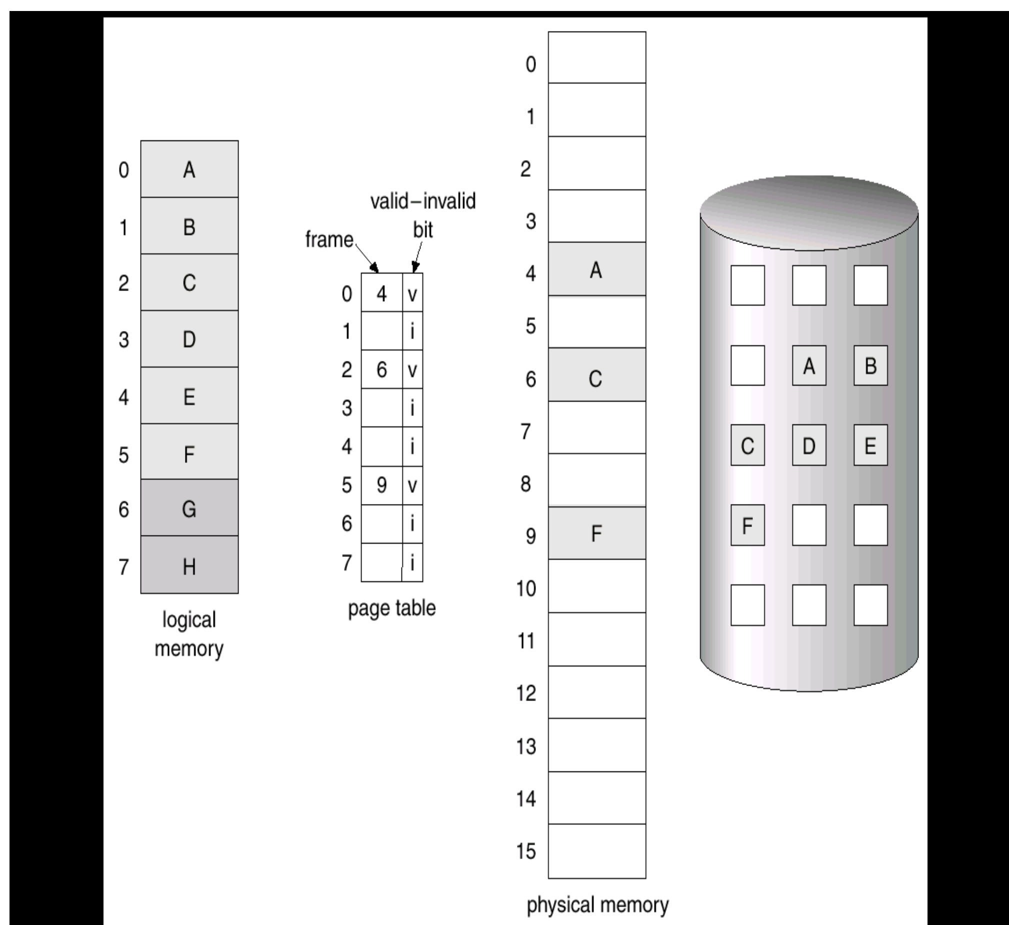
Một hệ thống phân trang theo yêu cầu tương tự một hệ thống phân trang với hoán vị (hình VIII-2). Các quá trình định vị trong bộ nhớ phụ (thường là đĩa). Khi chúng ta muốn thực thi một quá trình, chúng ta hoán vị nó vào bộ nhớ. Tuy nhiên, thay vì hoán vị toàn bộ quá trình ở trong bộ nhớ, chúng ta dùng một bộ hoán vị lười (lazy swapper). Bộ hoán vị lười không bao giờ hoán vị một trang vào trong bộ nhớ trừ khi trang đó sẽ được yêu cầu. Vì bây giờ chúng ta xem một quá trình như một chuỗi các trang hơn là một không gian địa chỉ liên tục có kích thước lớn, nên dùng hoán vị là không phù hợp về kỹ thuật. Một bộ hoán vị thao tác toàn bộ quá trình, ngược lại một bộ phân trang (pager) được quan tâm với các trang riêng rẻ của một quá trình. Do đó, chúng ta dùng bộ phân trang (hơn là bộ hoán vị) trong nối kết với phân trang theo yêu cầu.



Hình VIII-2 Chuyển bộ nhớ được phân trang tới không gian đĩa liên tục

Các khái niệm cơ bản

Với cơ chế này, chúng ta cần một số dạng phần cứng hỗ trợ để phân biệt giữa các trang ở trong bộ nhớ và các trang ở trên đĩa. Cơ chế bit hợp lệ-không hợp lệ có thể được dùng cho mục đích này. Tuy nhiên, thời điểm này khi bit được đặt "hợp lệ", giá trị này hiển thị rằng trang được tham chiếu tới là hợp lệ và ở đang trong bộ nhớ. Nếu một bit được đặt "không hợp lệ", giá trị này hiển thị rằng trang không hợp lệ (nghĩa là trang không ở trong không gian địa chỉ của quá trình) hoặc hợp lệ nhưng hiện đang ở trên đĩa. Mục từ bảng trang cho trang không ở trong bộ nhớ đơn giản được đánh dấu không hợp lệ, hay chứa địa chỉ của trang trên đĩa. Trường hợp này được mô tả trong hình VIII-3.



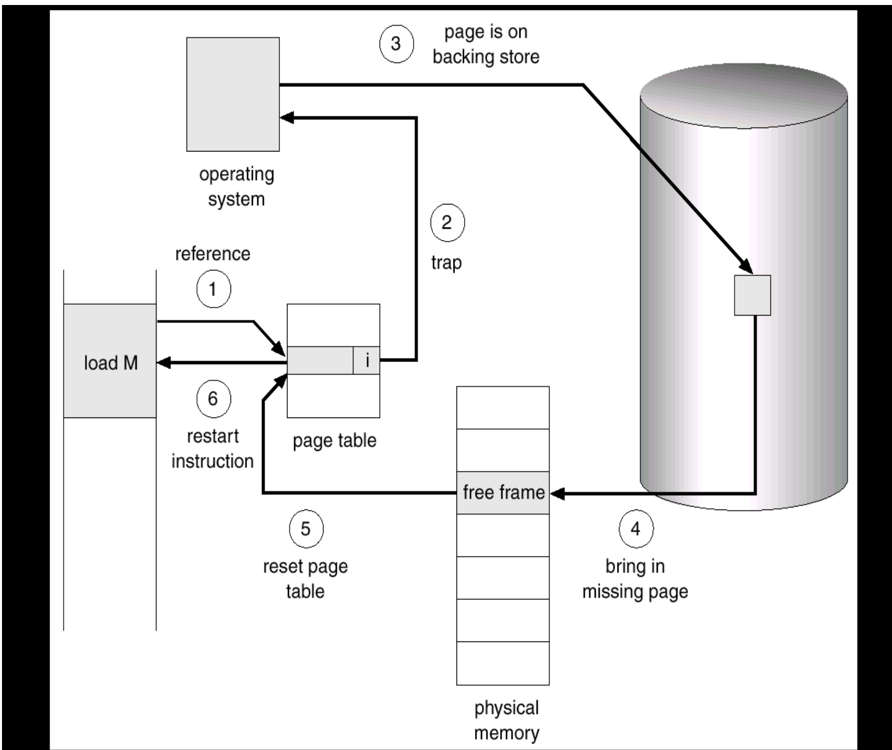
Hình VIII-3 Bảng trang khi một số trang không ở trong bộ nhớ chính

Chú ý rằng, đánh dấu một trang là "không hợp lệ" sẽ không có tác dụng nếu quá trình không bao giờ truy xuất trang đó. Do đó, nếu chúng ta đoán đúng và tất cả những trang thật sự cần đều ở trong bộ nhớ, quá trình sẽ chạy chính xác như khi chúng ta mang tất cả trang vào bộ nhớ. Trong khi quá trình thực thi và truy xuất trang đang định vị trong bộ nhớ, việc thực thi xử lý bình thường.

Nhưng điều gì xảy ra nếu quá trình cố gắng truy xuất trang mà trang đó không được mang vào bộ nhớ? Truy xuất một trang được đánh dấu là "không hợp lệ" gây ra một trap lỗi trang (page-fault trap). Phần cứng phân trang, dịch địa chỉ thông qua bảng trang, sẽ thông báo rằng bit không hợp lệ được đặt, gây ra một trap tới hệ điều hành. Trap này là kết quả lỗi của hệ điều hành mang trang được mong muốn vào bộ nhớ (trong một cố gắng tối thiểu chi phí chuyển đĩa và yêu cầu bộ nhớ) hơn là lỗi địa chỉ

không hợp lệ như kết quả của việc cố gắng dùng một địa chỉ bộ nhớ không hợp lệ (như một ký hiệu mảng không hợp lệ). Do đó, chúng ta phải sửa trường hợp sơ xuất này. Thủ tục cho việc quản lý lỗi trang này là không phức tạp (hình VIII-4).

1. Chúng ta kiểm tra bảng bên trong (thường được giữ với khối điều khiển quá trình) cho quá trình này, để xác định tham chiếu là truy xuất bộ nhớ hợp lệ hay không hợp lệ.
2. Nếu tham chiếu là không hợp lệ, chúng ta kết thúc quá trình. Nếu nó là hợp lệ, nhưng chúng ta chưa mang trang đó vào bộ nhớ, bây giờ chúng ta mang trang đó vào.
3. Chúng ta tìm khung trống (thí dụ, bằng cách mang một trang từ danh sách khung trống).
4. Chúng ta lập thời biểu thao tác đĩa để đọc trang mong muốn vào khung trang vừa mới được cấp phát.
5. Khi đọc đĩa hoàn thành, chúng ta sửa đổi bảng bên trong với quá trình và bảng trang để hiển thị rằng trang bây giờ ở trong bộ nhớ.
6. Chúng ta khởi động lại chỉ thị mà nó bị ngắt bởi trap địa chỉ không hợp lệ. Bây giờ quá trình có thể truy xuất trang mặc dù nó luôn ở trong bộ nhớ.



Hình VIII-4 Các bước quản lý lỗi trang

Vì chúng ta lưu trạng thái (thanh ghi, mã điều kiện, bộ đếm chỉ thị lệnh) của quá trình bị ngắt khi lỗi trang xảy ra, nên chúng ta có thể khởi động lại quá trình chính xác nơi và trạng thái, ngoại trừ trang mong muốn hiện ở trong bộ nhớ và có thể truy xuất. Trong cách này, chúng ta có thể thực thi một quá trình mặc dù các phần của nó chưa ở trong bộ nhớ. Khi quá trình cố gắng truy xuất các vị trí không ở trong bộ nhớ, phần cứng trap tới hệ điều hành (lỗi trang). Hệ điều hành đọc trang được yêu cầu vào bộ nhớ và khởi động lại quá trình như thể trang luôn ở trong bộ nhớ.

Trong trường hợp xấu nhất, chúng ta bắt đầu thực thi một quá trình với không trang nào ở trong bộ nhớ. Khi hệ điều hành đặt con trỏ chỉ thị lệnh tới chỉ thị đầu tiên của quá trình. Tuy nhiên, chỉ thị này ở trên trang không nằm trong bộ nhớ, quá trình lập tức báo lỗi đối với trang đó. Sau khi trang được mang vào trong bộ nhớ, quá trình tiếp tục thực thi, báo lỗi khi cần cho tới khi mỗi trang nó cần ở trong bộ nhớ. Tại thời điểm đó, nó có thể thực thi với không có lỗi nào nữa. Cơ chế này là thuần phân trang yêu

cầu (pure demand paging): không bao giờ mang trang vào bộ nhớ cho tới khi nó được yêu cầu.

Về lý thuyết, một số quá trình có thể truy xuất nhiều trang mới của bộ nhớ với mỗi sự thực thi chỉ thị (một trang cho một chỉ thị và nhiều trang cho dữ liệu), có thể gây ra lỗi nhiều trang trên chỉ thị. Trường hợp này sẽ dẫn đến năng lực thực hiện hệ thống không thể chấp nhận. May thay, phân tích các quá trình thực thi thể hiện rằng hành vi này là không hoàn toàn xảy ra. Các chương trình có khuynh hướng tham chiếu cục bộ dẫn đến năng lực phù hợp từ phân trang yêu cầu.

Phần cứng hỗ trợ phân trang theo yêu cầu là tương tự như phần cứng phân trang và hoán vị.

- Bảng trang: bảng này có khả năng đánh dấu mục từ không hợp lệ thông qua bit hợp lệ-không hợp lệ hay giá trị đặc biệt của các bit bảo vệ
- Bộ nhớ phụ: bộ nhớ này quản lý các trang không hiện diện trong bộ nhớ chính. Bộ nhớ phụ thường là đĩa tốc độ cao. Nó được xem như là thiết bị hoán vị và phần đĩa được dùng cho mục đích này được gọi là không gian hoán vị.

Ngoài sự hỗ trợ phần cứng này, phần mềm có thể xem xét được yêu cầu. Ràng buộc kiến trúc phải được áp đặt. Ràng buộc quan trọng được yêu cầu là có thể khởi động lại bất cứ chỉ thị nào sau khi lỗi trang. Trong hầu hết các trường hợp, yêu cầu này là dễ dàng thoả mãn. Lỗi trang có thể xảy ra tại bất cứ tham chiếu bộ nhớ nào. Nếu lỗi trang xảy ra trên việc lấy chỉ thị, chúng ta có thể khởi động lại bằng cách lấy lại chỉ thị. Nếu lỗi trang xảy ra trong khi chúng ta đang lấy một toán hạng, chúng ta phải lấy và giải mã lại chỉ thị, và sau đó lấy toán hạng.

Năng lực của phân trang theo yêu cầu

Phân trang theo yêu cầu có thể có một ảnh hưởng lớn trên năng lực của một hệ thống máy tính. Để thấy tại sao, chúng ta tính thời gian truy xuất

hiệu quả (effective access time) cho bộ nhớ được phân trang theo yêu cầu. Đối với hầu hết các hệ thống máy tính, thời gian truy xuất bộ nhớ, được ký hiệu ma, nằm trong khoảng từ 10 đến 200 nano giây. Với điều kiện là chúng ta không có lỗi trang, thời gian truy xuất hiệu quả là bằng với thời gian truy xuất bộ nhớ. Tuy nhiên, nếu lỗi trang xảy ra, trước hết chúng ta phải đọc trang tương ứng từ đĩa và sau đó truy xuất từ mong muốn.

Gọi p là xác suất của lỗi trang ($0 \le p \le 1$). Chúng ta mong đợi p gần bằng 0; nghĩa là chỉ có một vài lỗi trang. Thời gian truy xuất hiệu quả là:

Thời gian truy xuất hiệu quả = (1 – p) x ma + p x thời gian lỗi trang

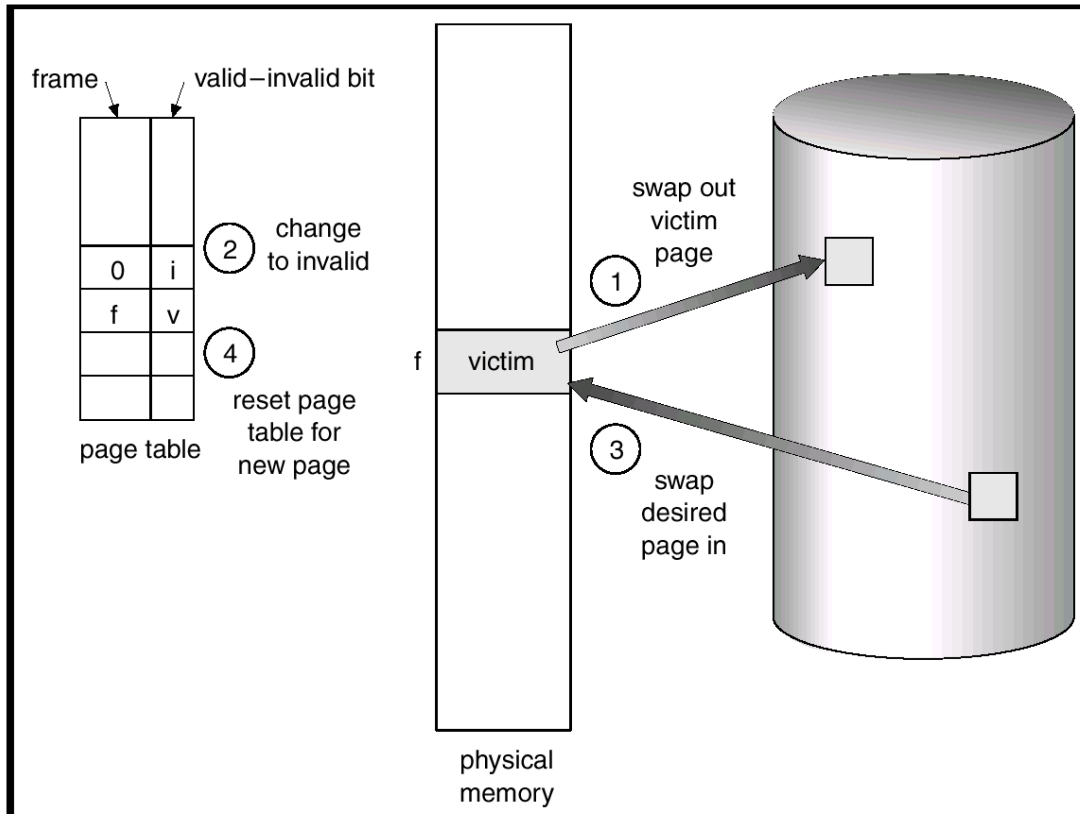
Để tính toán thời gian truy xuất hiệu quả, chúng ta phải biết phải mất bao lâu để phục vụ một lỗi trang. Để duy trì ở mức độ chấp nhận được sự chậm trễ trong hoạt động của hệ thống do phân trang, cần phải duy trì tỷ lệ phát sinh lỗi trang thấp.

Thay thế trang

Thay thế trang thực hiện tiếp cận sau. Nếu không có khung trống, chúng ta tìm một khung hiện không được dùng và giải phóng nó. Khi chúng ta giải phóng một khung bằng cách viết nội dung của nó tới không gian hoán vị và thay đổi bảng trang (và các bảng trang khác) để hiển thị rằng trang không còn ở trong bộ nhớ (hình VIII-5). Bây giờ chúng ta có thể dùng khung được giải phóng để quản lý trang cho quá trình bị lỗi. Chúng ta sửa đổi thủ tục phục vụ lỗi trang để chứa thay thế trang:

1. Tìm vị trí trang mong muốn trên đĩa
2. Tìm khung trang trống
    - Nếu có khung trống, dùng nó.
    - Nếu không có khung trống, dùng một giải thuật thay thế trang để chọn khung “nạn nhân”
    - Viết trang “nạn nhân” tới đĩa; thay đổi bảng trang và khung trang tương ứng.

3. Đọc trang mong muốn vào khung trang trống; thay đổi bảng trang và khung trang.
4. Khởi động lại quá trình.



Hình VIII-5 Thay thế trang

Chúng ta phải giải quyết hai vấn đề chính để cài đặt phân trang theo yêu cầu: chúng ta phát triển giải thuật cấp phát khung và giải thuật thay thế trang. Nếu chúng ta có nhiều quá trình trong bộ nhớ, chúng ta phải quyết định bao nhiêu khung cấp phát tới quá trình. Ngoài ra, khi thay thế trang được yêu cầu, chúng ta phải chọn các khung để được thay thế. Thiết kế các giải thuật hợp lý để giải quyết vấn đề này là một tác vụ quan trọng vì nhập/xuất đĩa là rất đắt. Thậm chí một cải tiến nhỏ trong các phương pháp phân trang theo yêu cầu sinh ra một lượng lớn năng lực hệ thống.

Có nhiều giải thuật thay thế trang khác nhau. Mỗi hệ điều hành có thể có cơ chế thay thế của chính nó. Chúng ta chọn một giải thuật thay thế trang như thế nào? Thông thường, chúng ta muốn một giải thuật tỉ lệ lỗi trang nhỏ nhất.

Chúng ta đánh giá một giải thuật bằng cách chạy nó trên một chuỗi các tham chiếu bộ nhớ cụ thể và tính số lượng lỗi trang. Chuỗi các tham chiếu bộ nhớ được gọi là chuỗi tham chiếu. Chúng ta có thể phát sinh chuỗi tham chiếu giả tạo (thí dụ, bằng bộ phát sinh số ngẫu nhiên). Chọn lựa sau đó tạo ra số lượng lớn dữ liệu (trên thứ tự 1 triệu địa chỉ trên giây). Để làm giảm số lượng dữ liệu này, chúng ta có hai cách

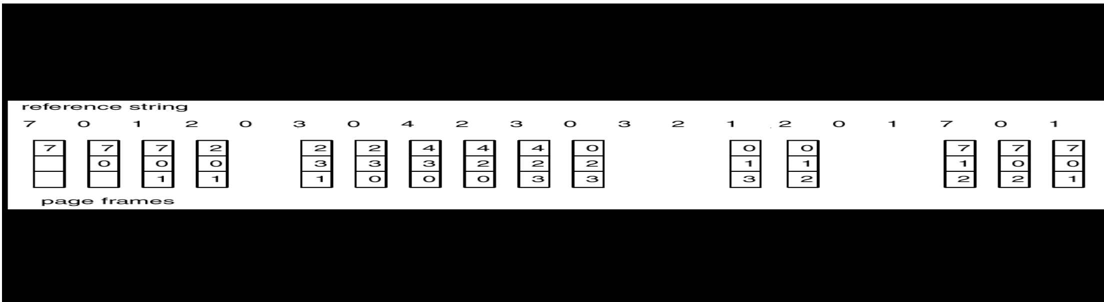
Cách thứ nhất, đối với kích thước trang được cho (và kích thước trang thường được cố định bởi phần cứng hay hệ thống), chúng ta cần xét chỉ số trang hơn là toàn địa chỉ. Cách thứ hai, nếu chúng ta có một tham chiếu tới trang p, thì bất cứ những tham chiếu tức thì theo sau tới trang p sẽ không bao giờ gây lỗi trang. Trang p sẽ ở trong bộ nhớ sau khi tham chiếu đầu tiên; các tham chiếu theo sau tức thì sẽ không bị lỗi.

Thay thế trang FIFO

Giải thuật thay thế trang đơn giản nhất là giải thuật FIFO. Giải thuật này gắn với mỗi trang thời gian khi trang đó được mang vào trong bộ nhớ. Khi một trang phải được thay thế, trang cũ nhất sẽ được chọn. Chú ý rằng, nó không yêu cầu nghiêm ngặt để ghi thời gian khi trang được mang vào. Chúng ta có thể tạo một hàng đợi FIFO để quản lý tất cả trang trong bộ nhớ. Chúng ta thay thế trang tại đầu hàng đợi. Khi trang được mang vào bộ nhớ, chúng ta chèn nó vào đuôi của hàng đợi.

Cho một thí dụ về chuỗi tham khảo, 3 khung của chúng ta ban đầu là rỗng. 3 tham khảo đầu tiên (7, 0, 1) gây ra lỗi trang và được mang vào các khung rỗng này. Tham khảo tiếp theo (2) thay thế trang 7, vì trang 7 được mang vào trước. Vì 0 là tham khảo tiếp theo và 0 đã ở trong bộ nhớ rồi, chúng ta không có lỗi trang cho tham khảo này. Tham khảo đầu tiên tới 3 dẫn đến trang 0 đang được thay thế vì thế nó là trang đầu tiên của 3 trang

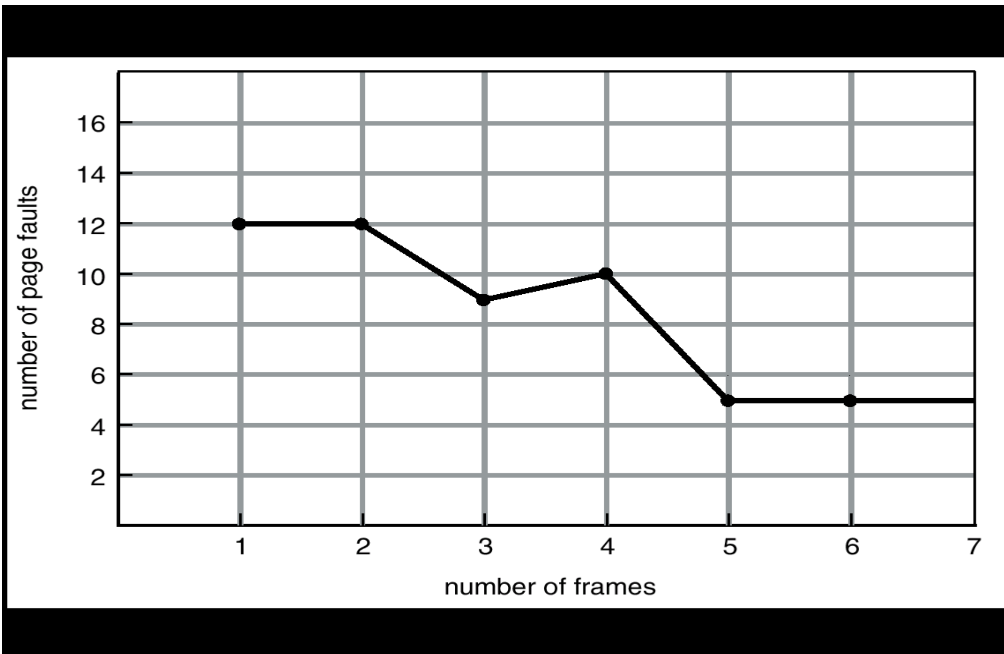
trong bộ nhớ (0, 1, 2) để được mang vào. Bởi vì thay thế này, tham khảo tiếp theo, tới 0, sẽ bị lỗi. Sau đó, trang 1 được thay thế bởi trang 0. Quá trình này tiếp tục như được hiển thị trong hình VIII-6. Mỗi khi một lỗi xảy ra, chúng ta hiển thị các trang ở trong 3 khung của chúng ta. Có 15 lỗi cả thảy.



Hình VIII-6 giải thuật thay thế trang FIFO

Giải thuật thay thế trang FIFO rất dễ hiểu và lập trình. Tuy nhiên, năng lực của nó không luôn tốt. Trang được cho để thay thế có thể là trang chức nhiều dữ liệu cần thiết, thường xuyên được sử dụng nên được nạp sớm, do vậy khi chuyển ra bộ nhớ phụ sẽ nhanh chóng gây ra lỗi trang.

Để hiển thị các vấn đề có thể phát sinh với giải thuật thay thế trang FIFO, chúng ta xem xét chuỗi tham khảo sau: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5. Hình VIII-7 hiển thị đường cong lỗi trang khi so sánh với số khung sẵn dùng. Chúng ta chú ý rằng số lượng lỗi cho 4 khung (10) là lớn hơn số lượng lỗi cho 3 khung (9). Hầu hết các kết quả không mong đợi này được gọi là sự nghịch lý Belady; đối với một số giải thuật thay thế trang, tỉ lệ lỗi trang có thể tăng khi số lượng khung được cấp phát tăng. Chúng ta sẽ mong muốn rằng cho nhiều bộ nhớ hơn tới một quá trình sẽ cải tiến năng lực của nó. Trong một vài nghiên cứu trước đây, các nhà điều tra đã kết luận rằng giả thuyết này không luôn đúng. Sự không bình thường của Belady được phát hiện như là một kết quả.



Hình VIII-7 Đường cong lỗi trang cho thay thế FIFO trên chuỗi tham khảo

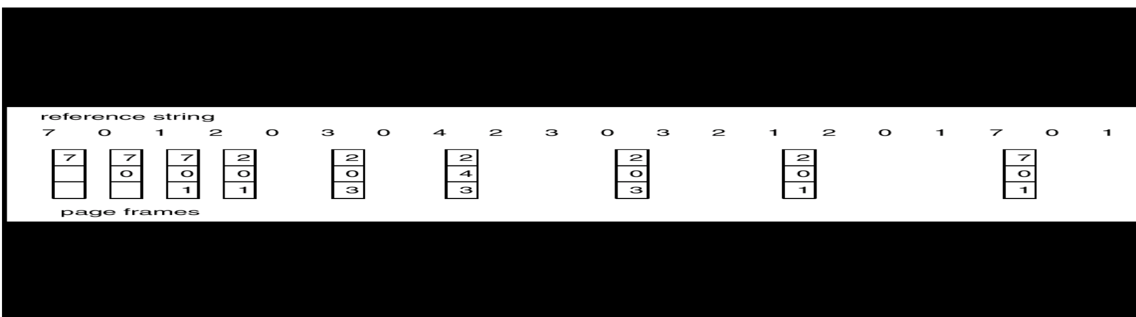
Thay thế trang tối ưu hoá

Kết quả phát hiện sự nghịch lý của Belady là tìm ra một giải thuật thay thế trang tối ưu. Giải thuật thay thế trang tối ưu có tỉ lệ lỗi trang thấp nhất trong tất cả các giải thuật và sẽ không bao giờ gặp phải sự nghịch lý của Belady. Giải thuật như thế tồn tại và được gọi là OPT hay MIN. Nó đơn giản là: thay thế trang mà nó không được dùng cho một khoảng thời gian lâu nhất. Sử dụng giải thuật thay thế trang đảm bảo tỉ lệ lỗi trang nhỏ nhất có thể cho một số lượng khung cố định.

Thí dụ, trên một chuỗi tham khảo mẫu, giải thuật thay thế trang tối ưu sẽ phát sinh 9 lỗi trang, như được hiển thị trong hình VIII-8. 3 tham khảo đầu tiên gây ra lỗi điền vào 3 khung trống. Tham khảo tới trang 2 thay thế trang 7 vì 7 sẽ không được dùng cho tới khi tham khảo 18, trái lại trang 0 sẽ được dùng tại 5 và trang 1 tại 14. Tham khảo tới trang 3 thay thế trang 1 khi trang 1 sẽ là trang cuối cùng của 3 trang trong bộ nhớ được tham khảo lần nữa. Với chỉ 9 lỗi trang, thay thế tối ưu là tốt hơn

nhiều giải thuật FIFO, có 15 lỗi. (Nếu chúng ta bỏ qua 3 lỗi đầu mà tất cả giải thuật phải gặp thì thay thế tối ưu tốt gấp 2 lần thay thế FIFO.) Thật vậy, không có giải thuật thay thế nào có thể xử lý chuỗi tham khảo trong 3 khung với ít hơn 9 lỗi.

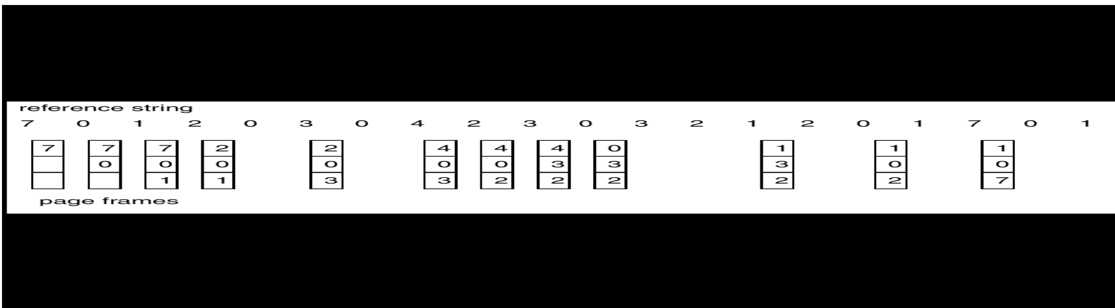
Tuy nhiên, giải thuật thay thế trang tối ưu là khó cài đặt vì nó yêu cầu kiến thức tương lai về chuỗi tham khảo. Do đó, giải thuật tối ưu được dùng chủ yếu cho nghiên cứu so sánh. Thí dụ, nó có thể có ích để biết rằng, mặc dù một giải thuật không tối ưu nhưng nó nằm trong 12.3% của tối ưu là tệ, và trong 4.7% là trung bình.



Hình VIII-8 giải thuật thay thế trang tối ưu

Thay thế trang LRU

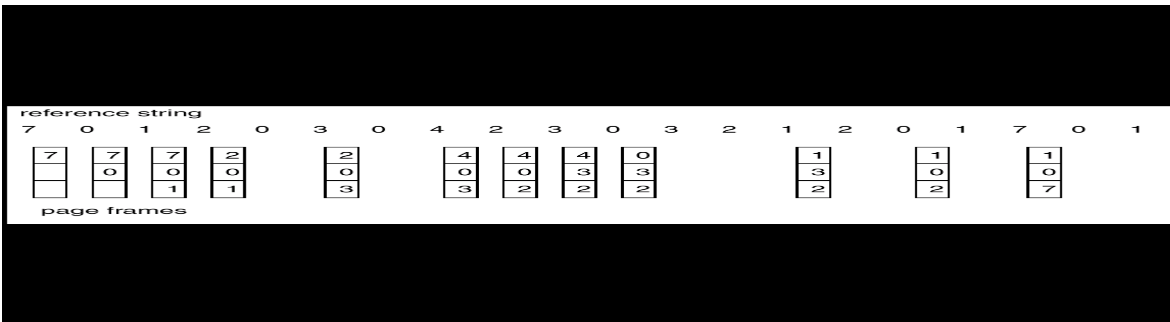
Nếu giải thuật tối ưu là không khả thi, có lẽ một xấp xỉ giải thuật tối ưu là có thể. Sự khác biệt chủ yếu giữa giải thuật FIFO và OPT là FIFO dùng thời gian khi trang được mang vào bộ nhớ; giải thuật OPT dùng thời gian khi trang được sử dụng. Nếu chúng ta sẽ dụng quá khứ gần đây như một xấp xỉ của tương lai gần thì chúng ta sẽ thay thế trang mà nó không được dùng cho khoảng thời gian lâu nhất (hình VIII-9). Tiếp cận này là giải thuật ít được dùng gần đây nhất (least-recently-used (LRU)).



Hình VIII-9 giải thuật thay thế trang LRU

Thay thế trang LRU gắn với mỗi trang thời gian sử dụng cuối cùng của trang. Khi một trang phải được thay thế, LRU chọn trang không được dùng trong một khoảng thời gian lâu nhất. Chiến lược này là giải thuật thay thế trang tối ưu tìm kiếm lùi theo thời gian hơn là hướng tới. (gọi SR là trình tự ngược của chuỗi tham khảo S thì tỉ lệ lỗi trang cho giải thuật OPT trên S là tương tự như tỉ lệ lỗi trang cho giải thuật OPT trên SR. Tương tự, tỉ lệ lỗi trang đối với giải thuật LRU trên S là giống như tỉ lệ lỗi trang cho giải thuật LRU trên SR)

Kết quả ứng dụng thay thế LRU đối với chuỗi tham khảo điển hình được hiển thị trong hình VIII-10. Giải thuật LRU sinh ra 12 lỗi. 5 lỗi đầu tiên là giống như thay thế tối ưu. Tuy nhiên, khi tham chiếu tới trang 4 xảy ra thay thế LRU thấy rằng 3 khung trong bộ nhớ, trang 2 được dùng gần đây nhất. Trang được dùng gần đây nhất là trang 0, và chỉ trước khi trang 3 được dùng. Do đó, giải thuật LRU thay thế trang 2, không biết rằng trang 2 để được dùng. Sau đó, khi nó gây lỗi trang 2, giải thuật LRU thay thế trang 3, của 3 trang trong bộ nhớ {0, 3, 4} trang 3 ít được sử dụng gần đây nhất. Mặc dù vấn đề này nhưng thay thế LRU với 12 lỗi vẫn tốt hơn thay thế FIFO với 15.



Hình VIII-10 giải thuật thay thế trang

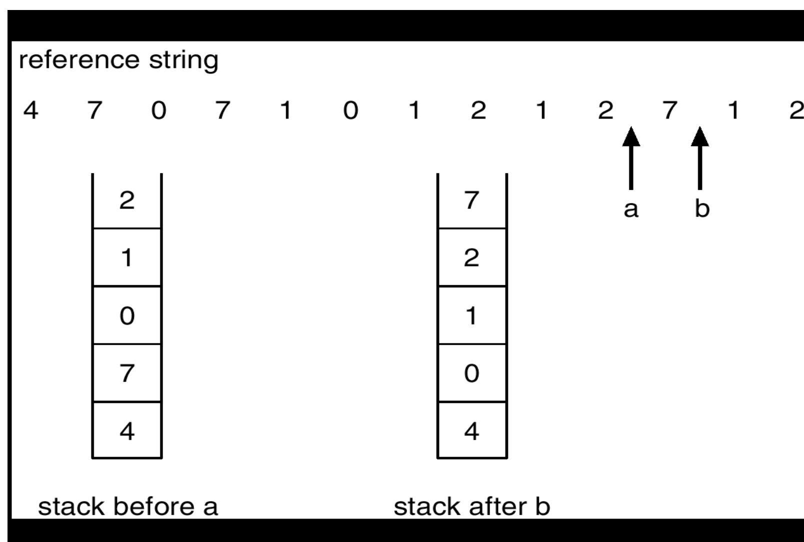
Chính sách LRU thường được dùng như giải thuật thay thế trang và được xem là tốt. Vấn đề chính là cách cài đặt thay thế LRU. Một giải thuật thay thế trang LRU có thể yêu cầu sự trợ giúp phần cứng. Vấn đề là xác định thứ tự cho các khung được định nghĩa bởi thời gian sử dụng gần nhất. Hai cách cài đặt khả thi là:

- Bộ đếm: trong trường hợp đơn giản nhất, chúng ta gắn mỗi mục từ bảng trang một trường số lần sử dụng và thêm CPU một đồng hồ luận lý hay bộ đếm. Đồng hồ được tăng cho mỗi tham khảo bộ nhớ. Bất cứ khi nào một tham khảo tới trang được thực hiện, các nội dung của thanh ghi đồng hồ được chép tới trường số lần sử dụng trong mục từ bảng trang cho trang đó. Trong cách này, chúng ta luôn có thời gian của tham khảo cuối cùng tới mỗi trang. Chúng ta thay thế trang với giá trị số lần sử dụng nhỏ nhất. Cơ chế này yêu cầu tìm kiếm bảng trang để tìm ra trang LRU và viết tới bộ nhớ (tới trường thời gian dùng trong bảng trang) cho mỗi truy xuất bộ nhớ. Số lần cũng phải được duy trì khi các bảng trang bị thay đổi (do định thời CPU). Vượt quá giới hạn của đồng hồ phải được xem xét.
- Ngăn xếp: một tiếp cận khác để cài đặt thay thế LRU là giữ ngăn xếp số trang. Bất cứ khi nào một trang được tham khảo, nó bị xoá từ ngăn xếp và đặt trên đỉnh. Trong cách này, đỉnh của ngăn xếp luôn là trang được dùng gần nhất và đáy là trang LRU (hình VIII-11). Vì các mục từ phải được xoá từ giữa ngăn xếp, nó được cài đặt tốt nhất bởi một danh sách được liên kết kép với con trỏ đầu và đuôi. Xoá một trang và đặt nó trên đỉnh của ngăn xếp sau đó yêu cầu thay đổi 6 con trỏ trong trường hợp xấu nhất. Mỗi cập nhật là ít chi phí hơn

nhưng không cần tìm một thay thế; con trỏ đuôi chỉ tới đáy của ngăn xếp là trang LRU. Tiếp cận này đặc biệt phù hợp cho cài đặt phần mềm hay vi mã của thay thế LRU.

Thay thế tối ưu hoá và LRU không gặp phải sự nghịch lý của Belady. Có một lớp giải thuật thay thế trang được gọi là giải thuật ngăn xếp, mà nó không bao giờ hiển thị sự nghịch lý Belady. Một giải thuật ngăn xếp là một giải thuật mà nó có thể được hiển thị rằng tập hợp trang trong bộ nhớ đối với n khung trang luôn là tập hợp con của tập hợp các trang mà nó ở trong bộ nhớ với n + 1 khung. Đối với thay thế LRU, tập hợp trang trong bộ nhớ là n trang được tham khảo gần đây nhất. Nếu số trang được gia tăng thì n trang này sẽ vẫn là những trang được tham khảo gần đây nhất và vì thế sẽ vẫn ở trong bộ nhớ.

Chú ý rằng cài đặt LRU sẽ có thể không có sự trợ giúp phần cứng ngoại trừ thanh ghi TLB. Cập nhật các trường đồng hồ hay ngăn xếp phải được thực hiện cho mỗi tham khảo bộ nhớ. Nếu chúng ta sử dụng ngắt cho mỗi tham khảo bộ nhớ, cho phép phần mềm cập nhật cấu trúc dữ liệu thì nó sẽ làm chậm mỗi tham khảo bộ nhớ gần 1 phần 10. Rất ít hệ thống có thể chịu cấp độ chi phí đó cho việc quản lý bộ nhớ.



Hình VIII-11 sử dụng ngăn xếp để ghi những tham khảo trang gần nhất

Giải thuật thay thế trang xấp xỉ LRU

Rất ít hệ thống máy tính cung cấp đầy đủ hỗ trợ phần cứng cho thay thế trang LRU. Một số hệ thống không cung cấp bất cứ sự hỗ trợ phần cứng và giải thuật thay thế trang khác (như giải thuật FIFO) phải được dùng. Tuy nhiên, nhiều hệ thống cung cấp một vài hỗ trợ trong dạng 1 bit tham khảo. Bit tham khảo cho một trang được đặt bởi phần cứng, bất cứ khi nào trang đó được tham khảo (đọc hay viết tới bất cứ bit nào trong trang). Các bit tham khảo gắn liền với mỗi mục từ trong bảng trang.

Ban đầu, tất cả bit được xoá (tới 0) bởi hệ điều hành. Khi một quá trình người dùng thực thi, bit được gán với mỗi trang được tham khảo được đặt (tới 1) bởi phần cứng. Sau thời gian đó, chúng có thể xác định trang nào được dùng và trang nào không được dùng bằng cách xem xét các bit tham khảo. Chúng ta không biết thứ tự sử dụng nhưng chúng ta biết trang nào được dùng và trang nào không được dùng. Thông tin thứ tự từng phần dẫn tới nhiều giải thuật thay thế trang xấp xỉ thay thế LRU.

Giải thuật các bit tham khảo phụ

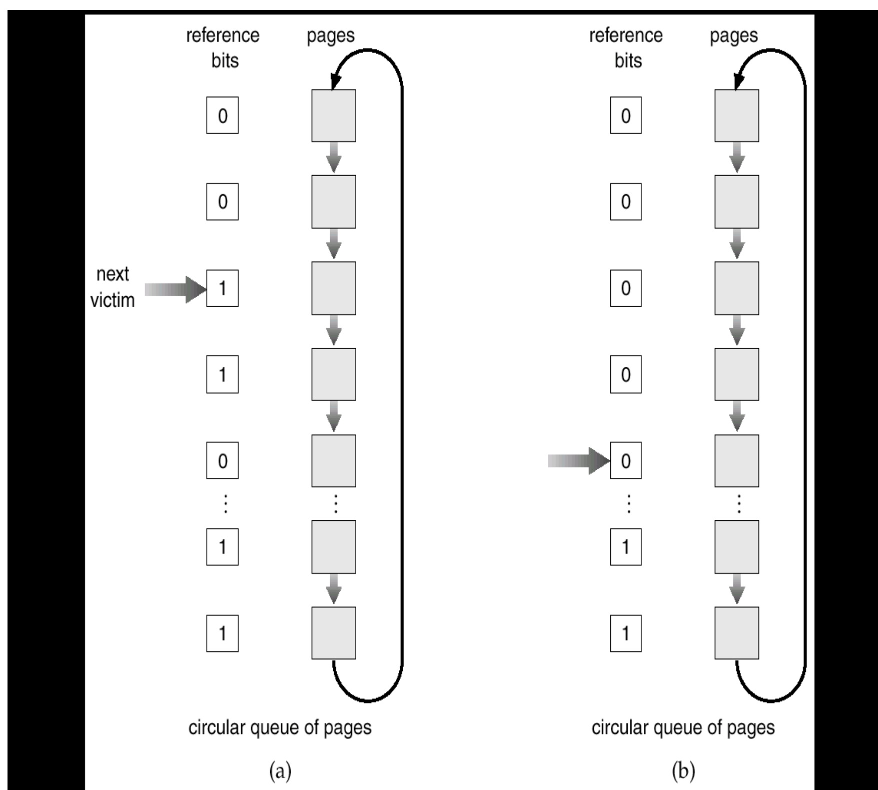
Chúng ta có thể nhận thêm thông tin thứ tự bằng cách ghi nhận các bit tham khảo tại những khoảng thời gian đều đặn. Chúng ta có thể giữ một byte cho mỗi trang trong một bảng nằm trong bộ nhớ. Tại những khoảng thời gian đều đặn (mỗi 100 mili giây), một ngắt thời gian chuyển điều khiển tới hệ điều hành. Hệ điều hành chuyển bit tham khảo cho mỗi trang vào bit có trọng số lớn nhất của byte, dịch các bit còn lại sang phải 1 bit. Xoá bit có trọng số thấp nhất. Thanh ghi dịch 8 bit có thể chứa lịch sử của việc sử dụng trang đối với 8 lần gần nhất. Nếu thanh ghi dịch chứa 00000000, thì trang không được dùng cho 8 thời điểm; một trang được dùng ít nhất một lần mỗi thời điểm sẽ có giá trị thanh ghi dịch là 11111111.

Một thanh ghi với giá trị thanh ghi lịch sử là 11000100 được dùng gần đây hơn một trang với 01110111. Nếu chúng ta thông dịch 8 bit này như số nguyên không dấu, trang với số thấp nhất là trang LRU và nó có thể được

thay thế. Tuy nhiên, các số này không đảm bảo duy nhất, chúng ta thay thế tất cả trang với giá trị nhỏ nhất hay dùng FIFO để chọn giữa chúng.

Dĩ nhiên, số lượng bit lịch sử có thể khác nhau và có thể được chọn (phụ thuộc phần cứng sẵn có) để thực hiện cập nhật nhanh nhất có thể. Trong trường hợp cực độ, số có thể được giảm về 0, chỉ bit tham khảo chính nó. Giải thuật này được gọi là giải thuật thay thế trang cơ hội thứ hai (second-chance page-replacement algorithm).

Giải thuật cơ hội thứ hai



Hình VIII-12 giải thuật thay thế trang cơ hội thứ hai

Giải thuật thay thế trang cơ hội thứ hai cơ bản là giải thuật thay thế FIFO. Tuy nhiên, khi một trang được chọn, chúng ta xét bit tham khảo của nó. Nếu giá trị bit này là 0, chúng ta xử lý để thay thế trang này. Tuy nhiên, nếu bit tham khảo được đặt tới 1, chúng ta cho trang đó một cơ hội

thứ hai và di chuyển để chọn trang FIFO kế tiếp. Khi một trang nhận cơ hội thứ hai, bit tham khảo của nó được xoá và thời gian đến của nó được đặt lại là thời gian hiện hành. Do đó, một trang được cho cơ hội thứ hai sẽ không được thay thế cho đến khi tất cả trang khác được thay thế (hay được cho cơ hội thứ hai). Ngoài ra, nếu một trang được dùng đủ thường xuyên để giữ bit tham khảo của nó được đặt, nó sẽ không bao giờ bị thay thế.

Một cách để cài đặt giải thuật cơ hội thứ hai như một hàng đợi vòng. Một con trỏ hiển thị trang nào được thay thế tiếp theo. Khi một khung được yêu cầu, con trỏ tăng cho tới khi nó tìm được trang với bit tham khảo 0. Khi nó tăng, nó xoá các bit tham khảo (hình VIII-12). Một khi trang nạn nhân được tìm thấy, trang được thay thế và trang mới được chèn vào hàng đợi vòng trong vị trí đó. Chú ý rằng, trong trường hợp xấu nhất khi tất cả bit được đặt, con trỏ xoay vòng suốt toàn hàng đợi, cho mỗi trang một cơ hội thứ hai. Thay thế cơ hội thứ hai trở thành thay thế FIFO nếu tất cả bit được đặt.

Giải thuật cơ hội thứ hai nâng cao

Chúng ta có thể cải tiến giải thuật cơ hội thứ hai bằng cách xem xét cả hai bit tham khảo và sửa đổi như một cặp được xếp thứ tự. Với hai bit này , chúng ta có 4 trường hợp có thể:

1. (0,0) không được dùng mới đây và không được sửa đổi-là trang tốt nhất để thay thế.
2. (0,1) không được dùng mới đây nhưng được sửa đổi-không thật tốt vì trang cần được viết ra trước khi thay thế.
3. (1,0) được dùng mới đây nhưng không được sửa đổi-nó có thể sẽ nhanh chóng được dùng lại.
4. (1,1) được dùng mới đây và được sửa đổi-trang có thể sẽ nhanh chóng được dùng lại và trang sẽ cần được viết ra đĩa trước khi nó có thể được thay thế.

Khi thay thế trang được yêu cầu, mỗi trang ở một trong bốn trường hợp. Chúng ta dùng cùng một cơ chế như giải thuật đồng hồ, nhưng thay vì

xem xét trang chúng ta đang trỏ tới có bit tham khảo được đặt tới 1 hay không, chúng ta xem xét trường hợp mà trang đó đang thuộc về. Chúng ta thay thế trang đầu tiên được gặp trong trường hợp thấp nhất không rỗng. Có thể chúng ta phải quét hàng đợi vòng nhiều lần trước khi chúng ta tìm một trang được thay thế.

Giải thuật này được dùng trong cơ chế quản lý bộ nhớ ảo của Macintosh. Sự khác nhau chủ yếu giữa giải thuật này và giải thuật đồng hồ đơn giản hơn là chúng ta cho tham khảo tới các trang đó mà chúng được sửa đổi để cắt giảm số lượng nhập/xuất được yêu cầu.

Thay thế trang dựa trên cơ sở đếm

Có nhiều giải thuật khác có thể được dùng để thay thế trang. Thí dụ, chúng ta có thể giữ bộ đếm số lần tham khảo đối với mỗi trang và phát triển hai cơ chế sau:

- Giải thuật thay thế trang được dùng ít thường xuyên nhất (the least frequently used (LFU) page-replacement algorithm) yêu cầu trang với số đếm nhỏ nhất được thay thế. Lý do cho sự chọn lựa này là trang được dùng nên có bộ đếm tham khảo lớn. Giải thuật này gặp phải trường hợp: trang được dùng nhiều trong quá trình khởi tạo nhưng không bao giờ được dùng lại. Vì nó được dùng nhiều nên nó có bộ đếm lớn và vẫn ở trong bộ nhớ mặc dù nó không còn cần nữa. Một giải pháp là dịch bộ đếm sang phải 1 bit tại khoảng thời gian đều đặn, hình thành một bộ đếm sử dụng trung bình giảm theo hàm mũ.
- Giải thuật thay thế trang được dùng thường xuyên nhất (the most frequently used (MFU) page-replacement algorithm) thay thế trang có giá trị đếm lớn nhất, nghĩa là trang được sử dụng nhiều nhất.

Cấp phát khung trang

Chúng ta cấp phát lượng bộ nhớ trống cố định giữa các quá trình khác nhau như thế nào? Nếu chúng ta có 93 khung trang trống và 2 quá trình, bao nhiêu khung trang mỗi quá trình sẽ nhận?

Trường hợp đơn giản nhất của bộ nhớ ảo là hệ thống đơn nhiệm. Xét một hệ thống đơn nhiệm với 128 KB bộ nhớ được hình thành từ các trang có kích thước 1 KB. Do đó, có 128 khung trang. Hệ điều hành có thể lấy 35 KB, còn lại 93 khung trang cho quá trình người dùng. Dưới thuần phân trang yêu cầu, tất cả 93 khung trang đầu tiên được đặt vào danh sách khung trống. Khi một quá trình người dùng bắt đầu thực thi, nó sinh ra một chuỗi lỗi trang. Những lỗi trang 93 đầu tiên nhận những khung trống từ danh sách khung trống. Khi danh sách khung trống hết, một giải thuật thay thế trang được dùng để chọn một trong 93 trang đang ở trong bộ nhớ để thay thế với trang thứ 94, …Khi một quá trình kết thúc, khung trang 93 một lần nữa được thay thế trên danh sách khung trang trống.

Có nhiều thay đổi trên chiến lược đơn giản này. Chúng ta có thể yêu cầu hệ điều hành cấp phát tất cả vùng đệm của nó và không gian bảng từ danh sách khung trống. Khi không gian này không được dùng bởi hệ điều hành, nó có thể được dùng để hỗ trợ phân trang người dùng. Chúng ta có thể cố gắng giữ 3 khung trang trống được dự trữ trên danh sách khung trang trống tại tất cả thời điểm. Do đó, khi lỗi trang xảy ra có một khung trống sẵn có đối với trang. Trong khi hoán vị trang xảy ra, một thay thế có thể được chọn, sau đó trang được viết tới đĩa khi quá trình người dùng tiếp tục thực thi.

Một thay đổi khác cũng có thể thực hiện trên chiến lược cơ bản là quá trình người dùng được cấp phát bất cứ khung trang nào trống.

Một vấn đề khác phát sinh khi phân trang yêu cầu được kết hợp với đa chương. Đa chương đặt hai hay nhiều quá trình trong bộ nhớ tại cùng một thời điểm.

Số khung trang tối thiểu

Những chiến lược cấp phát khung trang bị ràng buộc trong nhiều cách khác nhau. Chúng ta không thể cấp phát nhiều hơn toàn bộ số khung trang sẵn có (nếu không có chia sẻ trang). Chúng ta cũng cấp phát ít nhất số

khung trang tối thiểu. Chú ý, khi số khung trang được cấp phát tới mỗi quá trình giảm, tỉ lệ lỗi trang tăng, giảm việc thực thi quá trình.

Ngoài ra, năng lực thực hiện việc cấp phát ngoài mong muốn chỉ có một vài khung trang, có số khung trang tối thiểu phải được cấp phát. Số lượng tối thiểu. Số tối thiểu này được qui định bởi kiến trúc máy tính. Nhớ rằng, khi lỗi trang xảy ra trước khi chỉ thị thực thi hoàn thành, chỉ thị phải bắt đầu lại. Do đó, chúng ta phải có đủ khung trang để giữ tất cả trang khác nhau mà bất cứ chỉ thị đơn có thể tham khảo.

Thí dụ, xét một máy trong đó tất cả chỉ thị tham khảo bộ nhớ chỉ có một địa chỉ bộ nhớ. Do đó, chúng ta cần ít nhất một khung trang cho chỉ thị và một khung trang cho tham khảo bộ nhớ. Ngoài ra, nếu định địa chỉ gián tiếp cấp 1 được phép (thí dụ, một chỉ thị load trên trang 16 có thể tham khảo tới một địa chỉ bộ nhớ trên trang 0, mà nó tham khảo gián tiếp tới trang 23), thì phân trang yêu cầu ít nhất 3 khung trên quá trình. Điều gì có thể xảy ra nếu một quá trình chỉ có hai khung trang.

Các giải thuật cấp phát trang

Có hai tiếp cận:

1. Cấp phát cố định
    - Cấp phát công bằng: nếu có m khung trang và n quá trình, mỗi quá trình được cấp m/n khung trang
    - Cấp phát theo tỉ lệ: dựa vào kích thước của tiến trình để cấp phát số khung trang:
        - Gọi si = kích thước của bộ nhớ ảo cho quá trình pi
        - S = ∑ si
        - m = tổng số khung trang có thể sử dụng
        - Cấp phát ai khung trang tới quá trình pi: ai = (si / S) m
2. Cấp phát theo độ ưu tiên

Sử dụng ý tưởng cấp phát theo tỷ lệ, nhưng lượng khung trang cấp cho quá trình phụ thuộc vào độ ưu tiên của quá trình hơn là phụ thuộc kích thước quá trình

Nếu quá trình pi phát sinh lỗi trang, chọn một trong các khung trang của nó để thay thế, hoặc chọn một khung trang của quá trình khác với độ ưu tiên thấp hơn để thay thế.

- Thay thế trang toàn cục hay cục bộ

Có thể phân các thuật toán thay thế trang thành hai lớp chính:

- Thay thế toàn cục: khi lỗi trang xảy ra với một quá trình, chọn trang "nạn nhân" từ tập tất cả các khung trang trong hệ thống, bất kể khung trang đó đang được cấp phát cho một quá trình khác.
- Thay thế cục bộ: yêu cầu chỉ được chọn trang thay thế trong tập các khung trang được cấp cho quá trình phát sinh lỗi trang

Một khuyết điểm của giải thuật thay thế toàn cục là các quá trình không thể kiểm soát được tỷ lệ phát sinh lỗi trang của mình. Vì thế, tuy giải thuật thay thế toàn cục nhìn chung cho phép hệ thống có nhiều khả năng xử lý hơn, nhưng nó có thể dẫn hệ thống đến tình trạng trì trệ toàn bộ hệ thống (thrashing).

Trì trệ toàn hệ thống

Nếu một quá trình không có đủ các khung trang để chứa những trang cần thiết cho xử lý thì nó sẽ thường xuyên phát sinh lỗi trang và vì thế phải dùng đến rất nhiều thời gian sử dụng CPU để thực hiện thay thế trang. Một hoạt động phân trang như thế được gọi là sự trì trệ (thrashing). Một quá trình lâm vào trạng thái trì trệ nếu nó sử dụng nhiều thời gian để thay thế hơn là để xử lý.

Hiện tượng này ảnh hưởng nghiêm trọng đến hoạt động hệ thống, xét tình huống sau:

1. Hệ điều hành giám sát việc sử dụng CPU
2. Nếu hiệu suất sử dụng CPU quá thấp, hệ điều hành sẽ nâng mức độ đa chương bằng cách đưa thêm một quá trình mới vào hệ thống.
3. Hệ thống có thể sử dụng giải thuật thay thế toàn cục để chọn các trang nạn nhân thuộc một tiến trình bất kỳ để có chỗ nạp quá trình mới, có thể sẽ thay thế cả các trang của tiến trình đang xử lý hiện hành.
4. Khi có nhiều quá trình trong hệ thống hơn, thì một quá trình sẽ được cấp ít khung trang hơn và do đó phát sinh nhiều lỗi trang hơn.
5. Khi các quá trình phát sinh nhiều lỗi trang, chúng phải trải qua nhiều thời gian chờ các thao tác thay thế trang hoàn tất, lúc đó hiệu suất sử dụng CPU lại giảm.
6. Hệ điều hành lại quay trở lại bước 1.

Theo kịch bản trên đây, hệ thống sẽ lâm vào tình trạng luẩn quẩn của việc giải phóng các trang để cấp phát thêm khung trang cho một quá trình, và các quá trình khác lại thiếu khung trang..và các quá trình không thể tiếp tục xử lý. Đây chính là tình trạng trì trệ toàn bộ hệ thống. Khi tình trạng trì trệ này xảy ra, hệ thống gần như mất khả năng xử lý, tốc độ phát sinh lỗi trang tăng rất cao không công việc nào có thể kết thúc vì tất cả quá trình đều bận rộn với việc phân trang.

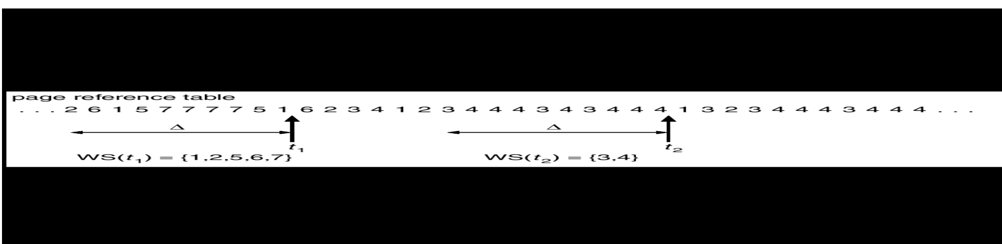
Để ngăn cản tình trạng trì trệ này xảy ra, cần phải cấp cho quá trình đủ các khung trang cần thiết để hoạt động. Vấn đề cần giải quyết là làm sao biết được quá trình cần bao nhiêu trang?

Mô hình cục bộ

Theo lý thuyết cục bộ thì khi một quá trình xử lý nó có khuynh hướng di chuyển từ nhóm trang cục bộ này đến nhóm trang cục bộ khác. Một nhóm trang cục bộ là một tập các trang đang được quá trình dùng đến trong một khoảng thời gian. Một chương trình thường bao gồm nhiều nhóm trang cục bộ khác nhau và chúng có thể giao nhau.

Mô hình tập làm việc

Mô hình tập làm việc (working set model) này dựa trên cơ sở lý thuyết cục bộ. Mô hình này sử dụng tham số Δ để định nghĩa cửa sổ cho tập làm việc. Giả sử, khảo sát Δ đơn vị thời gian (lần truy xuất trang) cuối cùng, tập các trang được quá trình truy xuất đến trong Δ lần truy cập cuối cùng được gọi là tập làm việc của quá trình tại thời điểm hiện tại. Nếu một trang đang được quá trình truy xuất tới, nó sẽ nằm trong tập làm việc nếu nó không sử dụng nữa, nó sẽ bị loại khỏi tập làm việc của quá trình sau Δ đơn vị thời gian kể từ lần truy xuất cuối cùng đến nó. Như vậy, tập làm việc chính là sự xấp xỉ của khái niệm nhóm trang cục bộ.



Hình VIII-13 Mô hình tập làm việc

Thuộc tính rất quan trọng của tập làm việc là kích thước của nó. Nếu tính toán kích thước tập làm việc WSSi, cho mỗi tiến trình trong hệ thống thì có thể xem:

D = ∑ WSSi

Với D là tổng số khung trang yêu cầu cho toàn hệ thống. Mỗi quá trình sử dụng các trang trong tập làm việc của nó, nghĩa là quá trình i yêu cầu WSSi khung trang. Nếu tổng số trang yêu cầu vượt quá tổng số trang có thể sử dụng trong hệ thống (D > m), thì sẽ xảy ra tình trạng trì trệ toàn bộ.

Dùng mô hình tập làm việc là đơn giản. Hệ điều hành kiểm soát tập làm việc của mỗi quá trình và cấp phát cho quá trình tối thiểu các khung trang để chứa đủ tập làm việc của nó. Nếu có đủ khung trang bổ sung thì quá

trình khác có thể được khởi tạo. Nếu tổng kích thước tập làm việc gia tăng vượt quá tổng số khung sẵn có, hệ điều hành chọn một quá trình để tạm dừng. Những trang của quá trình này được viết ra đĩa và các khung trang của nó được cấp phát lại cho quá trình khác. Quá trình được tạm dừng có thể khởi động lại sau đó.

Chiến lược tập làm việc ngăn chặn sự trì trệ trong khi giữ cấp độ đa chương cao nhất có thể. Do đó, nó tối ưu việc sử dụng CPU.

Khó khăn với mô hình tập làm việc này là giữ vết của tập làm việc. Cửa sổ tập làm việc là một cửa sổ di chuyển. Tại mỗi tham khảo bộ nhớ, một tham khảo mới xuất hiện khi một tham khảo trước đó kết thúc và tham khảo cũ nhất trở thành điểm kết thúc khác. Một trang ở trong tập làm việc nếu nó được tham khảo bất cứ nơi nào trong cửa sổ tập làm việc. Chúng ta có thể xem mô hình tập làm việc gần xấp xỉ với ngắt đồng hồ sau từng chu kỳ cố định và bit tham khảo.

Tần suất lỗi trang

Tần suất lỗi trang rất cao khiến tình trạng trì trệ hệ thống xảy ra. Khi tần suất lỗi trang quá cao, quá trình cần thêm một số khung trang. Ngược lại, khi tần suất quá thấp, quá trình có thể sở hữu nhiều khung trang hơn mức cần thiết. Có thể thiết lập một giá trị cận trên và cận dưới cho tần suất xảy ra lỗi trang và trực tiếp ước lượng và kiểm soát tần suất lỗi trang để ngăn chặn tình trạng trì trệ xảy ra:

- Nếu tần suất lỗi trang vượt quá cận trên, cấp cho quá trình thêm một khung trang
- Khi tần suất lỗi trang thấp hơn cận dưới, thu hồi bớt một khung trang từ quá trình.

Với chiến lược tập làm việc, chúng ta có thể có phải tạm dừng một quá trình. Nếu tỉ lệ lỗi trang tăng và không có trang nào trống, chúng ta phải chọn một số quá trình và tạm dừng nó. Sau đó, những khung trang được giải phóng sẽ được phân phối lại cho các quá trình với tỉ lệ lỗi trang cao.

Các vấn đề khác

Kích thước trang

Kích thước trang thông thường được xác định bởi phần cứng. Không có sự chọn lựa lý tưởng cho kích thước trang:

- Kích thước trang càng lớn thì kích thước bảng trang càng giảm
- Kích thước trang càng nhỏ thì cho phép tổ chức nhóm trang cục bộ tốt hơn và giảm sự phân mãnh trong
- Thời gian nhập xuất nhỏ khi kích thước trang lớn
- Kích thước trang nhỏ thì có thể giảm số lượng thao tác nhập xuất cần thiết vì có thể xác định các nhóm trang cục bộ chính xác hơn
- Kích thước trang lớn sẽ giảm tần xuất lỗi trang

Đa số các hệ thống chọn kích thước trang là 4 KB.

Cấu trúc chương trình

Về nguyên tắc, kỹ thuật phân trang theo yêu cầu được thiết kế nhằm giúp người dùng khỏi bận tâm đến việc sử dụng bộ nhớ một cách hiệu quả. Tuy nhiên, nếu hiểu rõ tổ chức bộ nhớ trong kỹ thuật phân trang, lập trình viên có thể giúp cho hoạt động của hệ thống tốt hơn với chương trình được xây dựng phù hợp.

Thí dụ, giả sử 1 trang có kích thước 128 bytes, một chương trình khởi tạo và gán giá trị mảng có kích thước 128x128 như sau:

```
Var A: array[1..128] of array [1..128] of byte;

For i:= 1 to 128 do

For j:=1 to 128 do

A[i][j]:=0;
```

Trong Pascal, C, PL/I, mảng trên đây được lưu trữ theo thứ tự dòng, mỗi dòng mảng chiếm một trang bộ nhớ, do đó tổng số lỗi trang phát sinh sẽ là 128.

Trong Fortran, mảng trên đây lại được lưu trữ theo thứ tự cột, do đó tổng số lỗi trang phát sinh sẽ là 128x128 = 1638.

Neo các trang trong bộ nhớ chính

Khi áp dụng kỹ thuật phân trang đôi lúc có nhu cầu "neo" trong bộ nhớ chính một số trang quan trọng hoặc thường được sử dụng hoặc không thể chuyển ra bộ nhớ phụ để bảo toàn dữ liệu.

Khi đó sử dụng thêm một bit khoá gán tương ứng cho từng khung trang. Một khung trang có bit khoá được đặt sẽ không bị chọn để thay thế.

Tóm tắt

Mong muốn có thể thực thi một quá trình có không gian địa chỉ luận lý lớn hơn không gian địa chỉ vật lý sẳn có. Người lập trình có thể làm một quá trình như thế có thể thực thi bằng cách cấu trúc lại nó dùng cơ chế phủ lắp, nhưng thực hiện điều này thường là một tác vụ lập trình khó. Bộ nhớ ảo là một kỹ thuật cho phép không gian địa chỉ luận lý được ánh xạ vào bộ nhớ vật lý nhỏ hơn. Bộ nhớ ảo cho phép những quá trình cực lớn được chạy và cũng cho phép cấp độ đa chương được gia tăng, tăng khả năng sử dụng CPU. Ngoài ra, nó giải phóng người lập trình ứng dụng từ việc lo lắng khả năng sẳn có của bộ nhớ.

Thuần phân trang theo yêu cầu mang vào một trang cho tới khi trang đó được tham khảo. Tham khảo đầu tiên gây ra lỗi trang tới hệ điều hành. Hệ điều hành xem xét bảng trang bên trong để xác định nơi trang được định vị trên vùng bộ nhớ phụ. Bảng trang được cập nhật để phản ánh sự thay đổi này, cho phép một quá trình chạy mặc dù toàn bộ hình ảnh bộ nhớ của nó không ở trong bộ nhớ chính. Khi tỉ lệ lỗi trang tương đối thấp, năng lực có thể chấp nhận.

Chúng ta có thể dùng phân trang theo yêu cầu để giảm số khung trang được cấp phát tới quá trình. Sắp xếp này có thể tăng cấp độ đa chương (cho phép nhiều quá trình sẵn sằng thực thi tại một thời điểm). Nó cũng cho phép các quá trình được thực thi mặc dù yêu cầu bộ nhớ vượt quá toàn bộ bộ nhớ vật lý sẵn có. Những quá trình như thế chạy trong bộ nhớ ảo.

Nếu tổng số yêu cầu bộ nhớ vượt quá bộ nhớ vật lý, thì nó cần thay thế trang từ bộ nhớ tới các khung trang trống cho những trang mới. Những giải thuật thay thế trang khác nhau được dùng. Thay thế trang FIFO là dễ dàng đối với chương trình nhưng gặp phải lỗi Belady. Thay thế trang tối ưu yêu cầu kiến thức tương lai. Thay thế LRU là xấp xỉ tối ưu nhưng nó rất khó cài đặt. Hầu hết các giải thuật thay thế trang như giải thuật cơ hội thứ hai là xấp xỉ thay thế LRU.

Ngoài ra đối với giải thuật thay thế trang, chính sách cấp phát khung trang được yêu cầu. Cấp phát có thể cố định, đề nghị thay thế trang cục bộ, hay động, đề nghị thay thế toàn cục. Mô hình tập làm việc cho rằng các quá trình thực thi trong các vị trí. Tập làm việc là tập các trang trong các vị trí hiện hành. Theo đó, mỗi quá trình nên được cấp phát đủ các khung cho tập làm việc hiện hành của nó.

Nếu một quá trình không có đủ bộ nhớ cho tập làm việc của nó, nó sẽ bị trì trệ. Cung cấp đủ khung cho mỗi quá trình để tránh trì trệ có thể yêu cầu quá trình hoán vị và định thời.

Ngoài ra, để yêu cầu chúng ta giải quyết các vấn đề chính của thay thế trang và cấp phát khung trang, thiết kế hợp lý hệ thống phân trang yêu cầu chúng ta xem xét kích thước trang, nhập/xuất, khoá, phân lại trang, tạo quá trình, cấu trúc chương trình, sự trì trệ,.. Bộ nhớ ảo có thể được xem như một cấp của cơ chế phân cấp trong các cấp lưu trữ trong hệ thống máy tính. Mỗi cấp có thời gian truy xuất, kích thước và tham số chi phí của chính nó.

Hệ thống tập tin
Mục đích Sau khi học xong chương này, người học nắm được những kiến thức sau: - Hiểu các khía cạnh khác nhau của tập tin và cấu trúc thư mục - Hiểu các cơ chế quản lý, kiểm soát, bảo vệ tập tin khi có nhiều người cùng truy xuất - Hiểu cách chia sẻ tập tin giữa nhiều quá trình, người dùng và máy tính

Giới thiệu

Đối với hầu hết người dùng, hệ thống tập tin là diện mạo dễ nhìn thấy nhất của hệ điều hành. Nó cung cấp cơ chế cho việc lưu trữ trực tuyến và truy xuất dữ liệu, chương trình của hệ điều hành và tất cả người dùng của hệ thống máy tính. Hệ thống tập tin chứa hai phần riêng biệt: tập hợp các tập tin (files), mỗi tập tin lưu trữ dữ liệu có liên quan và cấu trúc thư mục (directory structure) mà nó tổ chức và cung cấp thông tin về tất cả tập tin trong hệ thống. Một số hệ thống tập tin còn có thêm phần thứ ba , các phân khu (partitions) mà nó được dùng để tách rời tập hợp các thư mục lớn luận lý và vật lý.

Trong chương này chúng ta xét các khía cạnh khác nhau của tập tin và cấu trúc thư mục. Chúng ta cũng thảo luận các cách để quản lý việc bảo vệ tập tin (file protection), cần thiết khi nhiều người dùng truy xuất các tập tin và chúng ta muốn kiểm soát ai và cách gì truy xuất tập tin. Cuối cùng, chúng ta thảo luận việc chia sẻ giữa nhiều quá trình, người dùng, và máy tính.

Khái niệm tập tin

Các máy tính lưu trữ thông tin trên nhiều phương tiện lưu trữ khác nhau, như đĩa từ, băng từ, đĩa quang. Để hệ thống máy tính tiện dụng, hệ điều hành cung cấp một tầm nhìn luận lý không đổi của việc lưu trữ thông tin. Hệ điều hành trừu tượng từ các thuộc tính vật lý của các thiết bị lưu trữ của nó đến định nghĩa một đơn vị lưu trữ luận lý là tập tin (file). Tập tin được ánh xạ bởi hệ điều hành trên các thiết bị vật lý. Các thiết bị lưu trữ được dùng thường ổn định vì thế nội dung không bị mất khi mất điện hay khởi động lại hệ thống.

Một tập tin là một tập thông tin có liên quan được ghi trên thiết bị lưu trữ phụ. Từ quan điểm người dùng, một tập tin là phần nhỏ nhất của thiết bị lưu trữ phụ luận lý; nghĩa là dữ liệu không thể được viết tới thiết bị lưu trữ phụ trừ khi chúng ở trong một tập tin. Các tập tin dữ liệu có thể là số, chữ, ký tự số hay nhị phân. Các tập tin có thể có dạng bất kỳ như tập tin văn bản, hay có thể được định dạng không đổi. Thông thường, một tập tin là một chuỗi các bits, bytes, dòng hay mẫu tin,..được định nghĩa bởi người tạo ra nó. Do đó, khái niệm tập tin là cực kỳ tổng quát.

Thông tin trong một tập tin được định nghĩa bởi người tạo. Nhiều loại thông tin khác nhau có thể được lưu trữ trong một tập tin-chương trình nguồn, chương trình đối tượng, chương trình thực thi, dữ liệu số, văn bản, mẫu tin, hình ảnh đồ hoạ, âm thanh,..Một tập tin có một cấu trúc được định nghĩa cụ thể dựa theo loại của nó. Một tập tin văn bản là một chuỗi các ký tự được tổ chức thành những dòng. Một tập tin nguồn là một chuỗi các thủ tục và hàm, được tổ chức khi khai báo được theo sau bởi các câu lệnh có thể thực thi. Một tập tin đối tượng là một chuỗi các bytes được tổ chức thành các khối có thể hiểu được bởi bộ liên kết của hệ thống. Một tập tin có thể thực thi là một chuỗi các phần mã mà bộ nạp có thể mang vào bộ nhớ và thực thi.

Thuộc tính tập tin

Để tiện cho người dùng, một tập tin được đặt tên và được tham khảo bởi tên của nó. Một tên thường là một chuỗi các ký tự, thí dụ: example.c. Một số hệ thống có sự phân biệt giữa ký tự hoa và thường trong tên, ngược lại các hệ thống khác xem hai trường hợp đó là tương đương. Khi một tập tin được đặt tên, nó trở nên độc lập với quá trình, người dùng, và thậm chí với hệ thống tạo ra nó. Thí dụ, một người dùng có thể tạo tập tin example.c, ngược lại người dùng khác có thể sửa tập tin đó bằng cách xác định tên của nó. Người sở hữu tập tin có thể ghi tập tin tới đĩa mềm, gởi nó vào email hay chép nó qua mạng và có thể vẫn được gọi example.c trên hệ thống đích.

Một tập tin có một số thuộc tính khác mà chúng rất khác nhau từ một hệ điều hành này tới một hệ điều hành khác, nhưng điển hình chúng gồm:

- Tên (name): tên tập tin chỉ là thông tin được lưu ở dạng mà người dùng có thể đọc
- Định danh (identifier): là thẻ duy nhất, thường là số, xác định tập tin trong hệ thống tập tin; nó là tên mà người dùng không thể đọc
- Kiểu (type): thông tin này được yêu cầu cho hệ thống hỗ trợ các kiểu khác nhau
- Vị trí (location): thông tin này là một con trỏ chỉ tới một thiết bị và tới vị trí tập tin trên thiết bị đó.
- Kích thước (size): kích thước hiện hành của tập tin (tính bằng byte, word hay khối) và kích thước cho phép tối đa chứa trong thuộc tính này.
- Giờ (time), ngày (date) và định danh người dùng (user identification): thông tin này có thể được lưu cho việc tạo, sửa đổi gần nhất, dùng gần nhất. Dữ liệu này có ích cho việc bảo vệ, bảo mật, và kiểm soát việc dùng.

Thông tin về tất cả tập tin được giữ trong cấu trúc thư mục (directory) nằm trong thiết bị lưu trữ phụ. Điển hình, mục từ thư mục chứa tên tập tin và định danh duy nhất của nó. Định danh lần lượt xác định thuộc tính tập tin khác. Trong hệ thống có nhiều tập tin, kích thước của chính thư mục có thể là Mbyte. Bởi vì thư mục giống tập tin, phải bền, chúng phải được lưu trữ trên thiết bị và mang vào bộ nhớ khi cần.

Thao tác tập tin

Tập tin là kiểu dữ liệu trừu tượng. Để định nghĩa một tập tin hợp lý, chúng ta cần xem xét các thao tác có thể được thực hiện trên các tập tin. Hệ điều hành cung cấp lời gọi hệ thống để thực hiện các thao tác này

- Tạo tập tin: hai bước cần thiết để tạo một tập tin. Thứ nhất, không gian trong hệ thống tập tin phải được tìm cho tập tin. Thứ hai, một mục từ cho tập tin mới phải được tạo trong thư mục. Mục từ thư

mục ghi tên tập tin và vị trí trong hệ thống tập tin, và các thông tin khác.
- Mở: trước khi mở tập tin, quá trình phải mở nó. Mục tiêu của việc mở là cho phép hệ thống thiết lập một số thuộc tính và địa chỉ đĩa trong bộ nhớ để tăng tốc độ truy xuất.
- Đóng: khi chấm dứt truy xuất, thuộc tính và địa chỉ trên đĩa không còn dùng nữa, tập tin được đóng lại để giải phóng vùng nhớ.
- Ghi: để ghi một tập tin, chúng ta thực hiện lời gọi hệ thống xác định tên tập tin và thông tin được ghi tới tập tin. Với tên tập tin, hệ thống tìm thư mục để xác định vị trí của tập tin. Hệ thống phải giữ một con trỏ viết tới vị trí trong tập tin nơi mà thao tác viết tiếp theo sẽ xảy ra. Con trỏ viết phải được cập nhật bất cứ khi nào thao tác viết xảy ra.
- Chèn cuối: giống thao tác ghi nhưng dữ liệu luôn được ghi vào cuối tập tin
- Đọc: để đọc từ một tập tin, chúng ta dùng lời gọi hệ thống xác định tên tập tin và nơi (trong bộ nhớ) mà khối tiếp theo của tập tin được đặt. Thư mục được tìm mục từ tương ứng và hệ thống cần giữ con trỏ đọc tới vị trí trong tập tin nơi thao tác đọc tiếp theo xảy ra.
- Xoá: để xoá một tập tin, chúng ta tìm kiếm thư mục với tên tập tin được cho. Tìm mục từ tương ứng, giải phóng không gian tập tin để không gian này có thể dùng lại bởi tập tin khác và xoá mục từ thư mục.
- Tìm: thư mục được tìm mục từ tương ứng và vị trí con trỏ hiện hành được đặt tới giá trị được cho
- Lấy thuộc tính: lấy thuộc tính tập tin cho quá trình
- Đổi tên: thay đổi tên tập tin đã tồn tại

Các kiểu tập tin

Khi thiết kế một hệ thống tập tin, chúng ta luôn luôn xem xét hệ điều hành nên tổ chức và hỗ trợ các kiểu tập tin nào. Nếu hệ điều hành nhận biết kiểu của một tập tin, nó có thể thao tác trên tập tin đó trong các cách phù hợp.

Một kỹ thuật chung cho việc cài đặt các kiểu tập tin là chứa kiểu đó như một phần của tên tập tin. Tên tập tin được chia làm hai phần-tên và phần mở rộng, thường được ngăn cách bởi dấu chấm. Trong trường hợp này, người dùng và hệ điều hành có thể biết kiểu tập tin là gì từ tên.

Các hệ điều hành thường hỗ trợ các kiểu tập tin sau:

- Tập tin thường: là tập tin văn bản hay tập tin nhị phân chứa thông tin của người sử dụng
- Thư mục: là những tập tin hệ thống dùng để lưu giữ cấu trúc của hệ thống tập tin
- Tập tin có ký tự đặc biệt: liên quan đến nhập/xuất thông qua các thiết bị nhập/xuất tuần tự như màn hình, máy in,..
- Tập tin khối: dùng để truy xuất trên thiết bị đĩa

Cấu trúc tập tin

Các kiểu tập tin cũng có thể được dùng để hiển thị cấu trúc bên trong của một tập tin. Ngoài ra, các tập tin cụ thể phải phù hợp cấu trúc được yêu cầu để hệ điều hành có thể hiểu. Một số hệ điều hành mở rộng ý tưởng này thành tập hợp các cấu trúc tập tin được hỗ trợ hệ thống, với những tập hợp thao tác đặc biệt cho việc thao tác các tập tin với những cấu trúc đó.

Các hệ điều hành thường hỗ trợ ba cấu trúc tập tin thông dụng là:

- Không có cấu trúc: tập tin là một dãy tuần tự các byte
- Có cấu trúc: tập tin là một dãy các mẫu tin có kích thước cố định
- Cấu trúc cây: tập tin gồm một cây của những mẫu tin không cần thiết có cùng chiều dài, mỗi mẫu tin có một trường khoá giúp việc tìm kiếm nhanh hơn

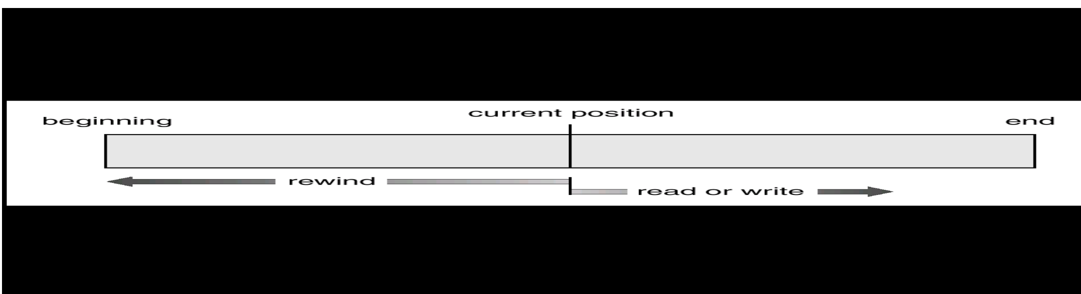
Các phương pháp truy xuất

Các tập tin lưu trữ thông tin. Khi nó được dùng, thông tin này phải được truy xuất và đọc vào bộ nhớ máy tính. Thông tin trong tập tin có thể được truy xuất trong nhiều cách.

Truy xuất tuần tự

Một phương pháp đơn giản nhất là truy xuất tuần tự. Thông tin trong tập tin được xử lý có thứ tự, một mẫu tin này sau mẫu tin kia. Chế độ truy xuất này là thông dụng nhất. Thí dụ, bộ soạn thảo và biên dịch thường truy xuất các tập tin trong cách thức này.

Nhóm các thao tác trên một tập tin là đọc và viết. Một thao tác đọc đọc phần tiếp theo của tập tin và tự động chuyển con trỏ tập tin để ghi vết vị trí nhập/xuất. Tương tự, một thao tác viết chèn vào cuối tập tin và chuyển tới vị trí cuối của tài liệu vừa được viết (cuối tập tin mới). Trên một vài hệ thống, một tập tin như thế có thể được đặt lại tới vị trí bắt đầu và một chương trình có thể nhảy tới hay lùi n mẫu tin. Truy xuất tuần tự được mô tả như hình IX-1.



Hình IX-1 Truy xuất tập tin tuần tự

Truy xuất trực tiếp

Một phương pháp khác là truy xuất trực tiếp (hay truy xuất tương đối). Một tập tin được hình thành từ các mẫu tin luận lý có chiều dài không đổi. Các mẫu tin này cho phép người lập trình đọc và viết các mẫu tin

nhanh chóng không theo thứ tự. Phương pháp truy xuất trực tiếp dựa trên mô hình đĩa của tập tin, vì đĩa cho phép truy xuất ngẫu nhiên tới bất cứ khối tập tin. Để truy xuất trực tiếp, tập tin được hiển thị như một chuỗi các khối hay mẫu tin được đánh số. Tập tin truy xuất trực tiếp cho phép các khối bất kỳ được đọc hay viết. Do đó, chúng ta có thể đọc khối 14, sau đó đọc khối 53 và sau đó viết khối 7. Không có bất kỳ sự hạn chế nào trên thứ tự đọc hay viết cho một tập tin truy xuất trực tiếp

Các tập tin truy xuất trực tiếp được dùng nhiều cho truy xuất tức thời tới một lượng lớn thông tin. Cơ sở dữ liệu thường là loại này. Khi một truy vấn tập trung một chủ đề cụ thể, chúng ta tính khối nào chứa câu trả lời và sau đó đọc khối đó trực tiếp để cung cấp thông tin mong muốn.

Không phải tất cả hệ điều hành đều hỗ trợ cả hai truy xuất tuần tự và trực tiếp cho tập tin. Một số hệ thống cho phép chỉ truy xuất tập tin tuần tự; một số khác cho phép chỉ truy xuất trực tiếp. Một số hệ điều hành yêu cầu một tập tin được định nghĩa như tuần tự hay trực tiếp khi nó được tạo ra; như tập tin có thể được truy xuất chỉ trong một cách không đổi với khai báo của nó. Tuy nhiên, chúng ta dễ dàng mô phỏng truy xuất tuần tự trên tập tin truy xuất trực tiếp. Nếu chúng ta giữ một biến cp để xác định vị trí hiện tại thì chúng ta có thể mô phỏng các thao tác tập tin tuần tự như được hiển thị trong hình IX-2. Mặc dù, không đủ và không gọn để mô phỏng một tập tin truy xuất trực tiếp trên một tập tin truy xuất tuần tự.

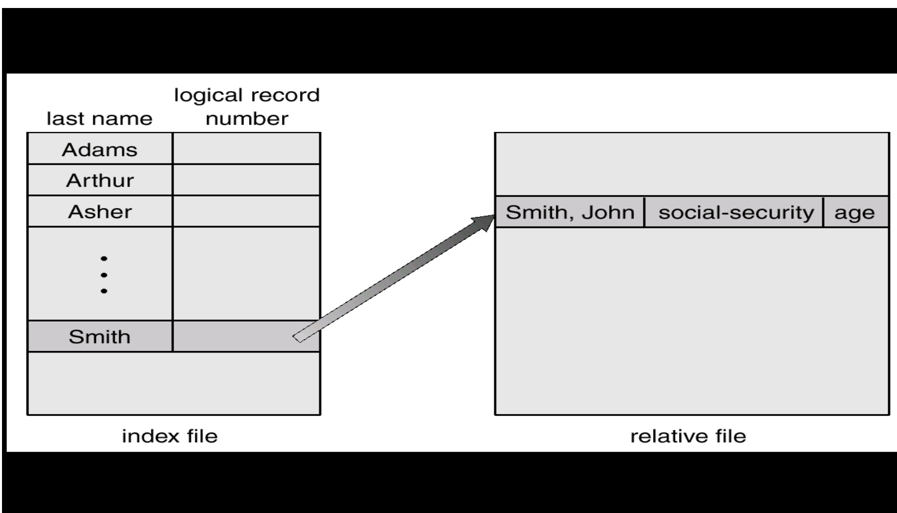
sequential access	implementation for direct access
reset	*cp = 0;*
read next	*read cp;* *cp = cp+1;*
write next	*write cp;* *cp = cp+1;*

Hình IX-2 Mô phỏng truy xuất tuần tự trên truy xuất trực tiếp

Các phương pháp truy xuất khác

Các phương pháp truy xuất khác có thể được xây dựng trên cơ sở của phương pháp truy xuất trực tiếp. Các phương pháp khác thường liên quan đến việc xây dựng chỉ mục cho tập tin. Chỉ mục chứa các con trỏ chỉ tới các khối khác. Để tìm một mẫu tin trong tập tin, trước hết chúng ta tìm chỉ mục và sau đó dùng con trỏ để truy xuất tập tin trực tiếp và tìm mẫu tin mong muốn.

Với những tập tin lớn, chỉ mục tập tin có thể trở nên quá lớn để giữ trong bộ nhớ. Một giải pháp là tạo chỉ mục cho tập tin chỉ mục. Tập tin chỉ mục chính chứa các con trỏ chỉ tới các tập tin chỉ mục thứ cấp mà nó chỉ tới các thành phần dữ liệu thật sự.



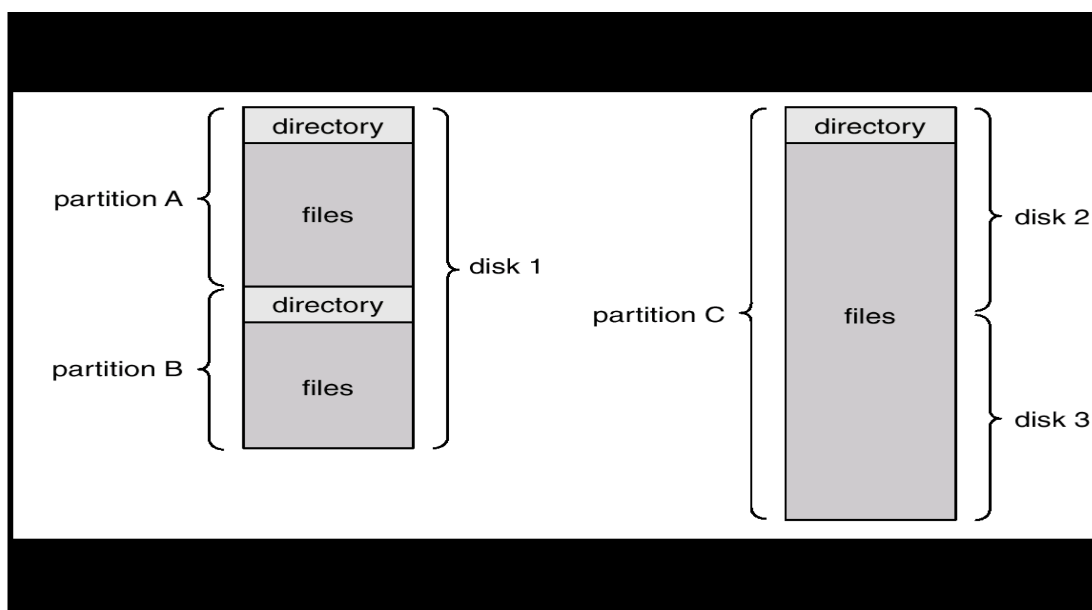
Hình IX-3 Thí dụ về chỉ mục và các tập tin liên quan

Cấu trúc thư mục

Các hệ thống tập tin của máy tính có thể rất lớn về số lượng. Một số hệ thống lưu trữ hàng triệu tập tin trên các terabytes đĩa. Để quản lý tất cả dữ liệu này, chúng ta cần tổ chức lại chúng. Việc tổ chức này thường được thực hiện hai phần.

Thứ nhất, đĩa được chia thành một hay nhiều phân khu (partition) hay phân vùng (volumes). Điển hình, mỗi đĩa trên hệ thống chứa ít nhất một phân khu. Phân khu này là cấu trúc cấp thấp mà các tập tin và thư mục định vị. Thỉnh thoảng các phân khu được dùng để cung cấp nhiều vùng riêng rẻ trong một đĩa, mỗi phân khu được xem như một thiết bị lưu trữ riêng, trái lại các hệ thống khác cho phép các phân khu có dung lượng lớn hơn một đĩa để nhóm các đĩa vào một cấu trúc luận lý và cấu trúc tập tin, và có thể bỏ qua hoàn toàn những vấn đề cấp phát không gian vật lý cho các tập tin. Cho lý do này, các phân khu có thể được xem như các đĩa ảo. Các phân khu cũng có thể lưu trữ nhiều hệ điều hành, cho phép hệ thống khởi động và chạy nhiều hơn một hệ điều hành.

Thứ hai, mỗi phân khu chứa thông tin về các tập tin trong nó. Thông tin này giữ trong những mục từ trong một thư mục thiết bị hay bảng mục lục phân vùng (volume table of contents). Thư mục thiết bị (được gọi đơn giản là thư mục) ghi thông tin-như tên, vị trí, kích thước và kiểu-đối với tất cả tập tin trên phân khu (như hình IX-4).



Hình IX-4 tổ chức hệ thống tập tin điển hình

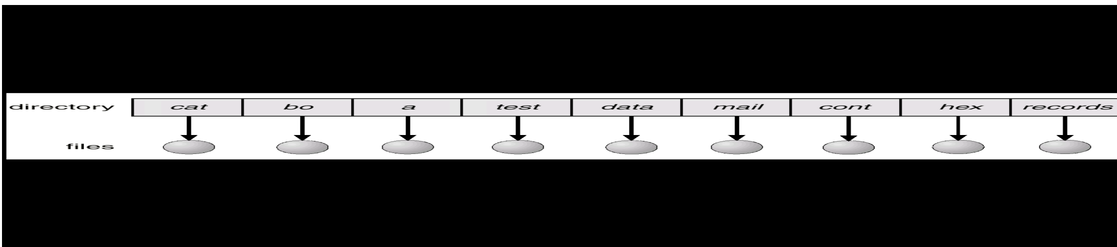
Thư mục có thể được hiển thị như một bảng danh biểu dịch tên tập tin thành các mục từ thư mục. Các thư mục có thể được tổ chức trong nhiều cách. Chúng ta muốn có thể chèn mục từ, xoá mục từ, tìm kiếm một mục từ và liệt kê tất cả mục từ trong thư mục. Trong phần này, chúng ta xem xét nhiều cơ chế định nghĩa cấu trúc luận lý của hệ thống thư mục. Khi xem xét một cấu trúc thư mục cụ thể, chúng ta cần nhớ các thao tác được thực hiện trên một thư mục.

- Tìm kiếm tập tin: chúng ta cần tìm trên cấu trúc thư mục để xác định mục từ cho một tập tin cụ thể.
- Tạo tập tin: một tập tin mới cần được tạo và được thêm tới thư mục.
- Xoá tập tin: khi một tập tin không còn cần, chúng ta muốn xoá nó ra khỏi thư mục.
- Liệt kê thư mục: chúng ta có thể liệt kê các tập tin trong thư mục và nội dung của mục từ thư mục cho mỗi tập tin trong danh sách.
- Đổi tên tập tin: vì tên tập tin biểu diễn nội dung của nó đối với người dùng, tên có thể thay đổi khi nội dung hay việc sử dụng tập tin thay đổi. Đổi tên tập tin có thể cho phép vị trí của nó trong cấu trúc thư mục được thay đổi.
- Duyệt hệ thống tập tin: chúng ta muốn truy xuất mỗi thư mục và mỗi tập tin trong cấu trúc thư mục.

Chúng ta sẽ mô tả các cơ chế thông dụng nhất để định nghĩa cấu trúc luận lý của một thư mục.

Cấu trúc thư mục dạng đơn cấp

Cấu trúc thư mục đơn giản nhất là thư mục đơn cấp. Tất cả tập tin được chứa trong cùng thư mục như được hiển thị trong hình IX-5 dưới đây:



Hình IX-5 Thư mục đơn cấp

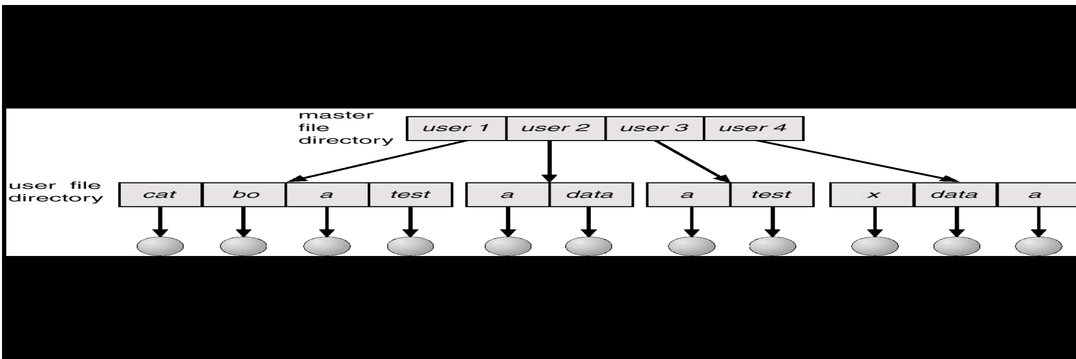
Tuy nhiên, thư mục đơn cấp có nhiều hạn chế khi số lượng tập tin tăng hay khi hệ thống có nhiều hơn một người dùng. Vì tất cả tập tin được chứa trong cùng thư mục, chúng phải có tên khác nhau. Nếu hai người dùng đặt tên tập tin dữ liệu của họ là test thì qui tắc tên duy nhất bị xung đột. Mặc dù các tên tập tin thường được chọn để phản ánh nội dung của tập tin, chúng thường bị giới hạn chiều dài. Hệ điều hành MS-DOS cho phép chỉ 11 ký tự cho tên; UNIX cho phép 255 ký tự.

Người dùng trên thư mục đơn cấp có thể gặp phải khó khăn để nhớ tên của tất cả tập tin, khi số tập tin tăng. Nếu trên một hệ thống máy tính có hàng trăm tập tin thì việc ghi lại vết của quá nhiều tập tin là một tác vụ nặng nề.

Cấu trúc thư mục dạng hai cấp

Một thư mục đơn cấp dẫn đến sự lẫn lộn giữa tên các tập tin của nhiều người dùng khác nhau. Giải pháp chuẩn là tạo một thư mục riêng cho mỗi người dùng.

Trong cấu trúc thư mục hai cấp, mỗi người dùng có thư mục tập tin riêng cho họ (user file directory-UFD). Mỗi UFD có một cấu trúc tương tự nhưng các danh sách chứa các tập tin của một người dùng. Khi công việc của người dùng bắt đầu hay người dùng đăng nhập, thư mục tập tin chính của hệ thống (master file directory) được tìm kiếm. MFD được lập chỉ mục bởi tên người dùng hay số tài khoản và mỗi mục từ chỉ tới UFD cho người dùng đó (như hình IX-6)



Hình IX-6 thư mục hai cấp

Khi người dùng tham khảo tới một tập tin cụ thể, chỉ UFD của chính người dùng đó được tìm kiếm. Do đó, các người dùng khác nhau có thể có các tập tin với cùng một tên, với điều kiện là tất cả tên tập tin trong mỗi UFD là duy nhất.

Để tạo một tập tin cho một người dùng, hệ điều hành chỉ tìm UFD của người dùng đó để xác định một tập tin khác cùng tên có tồn tại hay không. Để xóa một tập tin, hệ điều hành giữ lại việc tìm kiếm của nó tới UFD cục bộ; do đó, nó không thể xóa nhầm tập tin của người dùng khác có cùng tên.

Các thư mục người dùng phải được tạo và xóa khi cần thiết. Một chương trình hệ thống đặc biệt được chạy với tên người dùng hợp lý và thông tin tài khoản. Chương trình này tạo một UFD mới và thêm một mục từ cho nó tới MFD. Việc thực thi chương trình này có thể bị giới hạn bởi người quản trị hệ thống.

Mặc dù cấu trúc thư mục hai cấp giải quyết vấn đề xung đột tên nhưng nó cũng có những bất lợi. Cấu trúc này cô lập một người dùng từ người dùng khác. Việc cô lập này là lợi điểm khi các người dùng hoàn toàn độc lập nhau nhưng sẽ bất lợi khi các người dùng muốn hợp tác trên một số công việc và để truy xuất các tập tin của người dùng khác. Một số hệ thống đơn giản không cho phép tập tin người dùng cục bộ được truy xuất bởi người dùng khác.

Nếu truy xuất được cho phép, một người dùng phải có khả năng đặt tên một tập tin trong một thư mục của người dùng khác. Để đặt tên một tập tin xác định duy nhất trong thư mục hai cấp, chúng ta phải cho cả hai tên người dùng và tên tập tin. Một thư mục hai cấp có thể được xem như một cây hay ít nhất một cây đảo ngược hay có chiều cao bằng 2. Gốc của cây là UFD. Hậu duệ trực tiếp của nó là MFD. Hậu duệ của UFD là các tập tin. Các tập tin này là lá của cây. Xác định tên người dùng và tên tập tin định nghĩa đường dẫn trong cây từ gốc (MFD) tới một lá (tập tin xác định). Do đó, tên người dùng và tên tập tin định nghĩa tên đường dẫn. Mọi tập tin trong hệ thống có một đường dẫn. Để đặt tên một tập tin duy nhất người dùng phải biết tên đường dẫn của tập tin mong muốn.

Trường hợp đặc biệt xảy ra cho các tập tin hệ thống. Các chương trình này cung cấp một phần hệ thống như: bộ nạp, bộ hợp ngữ, bộ biên dịch, các thủ tục,..thường được định nghĩa như các tập tin. Khi các lệnh tương ứng được gọi tới hệ điều hành, các tập tin này được đọc bởi bộ nạp và được thực thi. Một số bộ thông dịch lệnh hoạt động bằng cách xem lệnh như là tên tập để nạp và thực thi. Với hệ thống thư mục được định nghĩa hiện tại, tên tập tin này được tìm kiếm trong UFD hiện hành. Một giải pháp cho vấn đề này là chép các tập tin hệ thống vào mỗi UFD. Tuy nhiên, chép tất cả tập tin hệ thống sẽ lãng phí lượng lớn không gian.

Giải pháp chuẩn là làm phức tạp thủ tục tìm kiếm một chút. Một thư mục người dùng đặc biệt được định nghĩa để chứa các tập tin hệ thống (thí dụ, user0). Bất cứ khi nào một tên tập tin được cho để được nạp, trước tiên hệ điều hành tự tìm thư mục người dùng cục bộ. Nếu không tìm thấy, hệ điều hành tự tìm trong thư mục người dùng đặc biệt này. Một chuỗi các thư mục được tìm khi một tập tin được đặt tên là đường dẫn tìm kiếm. Ý tưởng này có thể được mở rộng, đường dẫn tìm kiếm chứa danh sách các thư mục không giới hạn để tìm khi tên một lệnh được cho. Phương pháp này được dùng nhiều nhất trong UNIX và MS-DOS.

Cấu trúc thư mục dạng cây

Thư mục cấu trúc cây là trường hợp tổng quát của thư mục hai cấp. Sự tổng quát này cho phép người dùng tạo thư mục con và tổ chức các tập tin của họ. Thí dụ, hệ thống MS-DOS có cấu trúc cây. Thật vậy, một cây là cấu trúc thư mục phổ biến nhất. Cây có thư mục gốc. Mỗi tập tin trong hệ thống có tên đường dẫn duy nhất. Tên đường dẫn là đường dẫn từ gốc xuống tất cả thư mục con tới tập tin xác định.

Một thư mục (hay thư mục con) chứa tập hợp các tập tin hay thư mục con. Một thư mục đơn giản là tập tin nhưng nó được đối xử trong một cách đặc biệt. Tất cả thư mục có cùng định dạng bên trong. Một bit trong mỗi mục từ thư mục định nghĩa mục từ như một tập tin (0) hay như một thư mục con (1). Các lời gọi hệ thống đặc biệt được dùng để tạo và xoá thư mục.

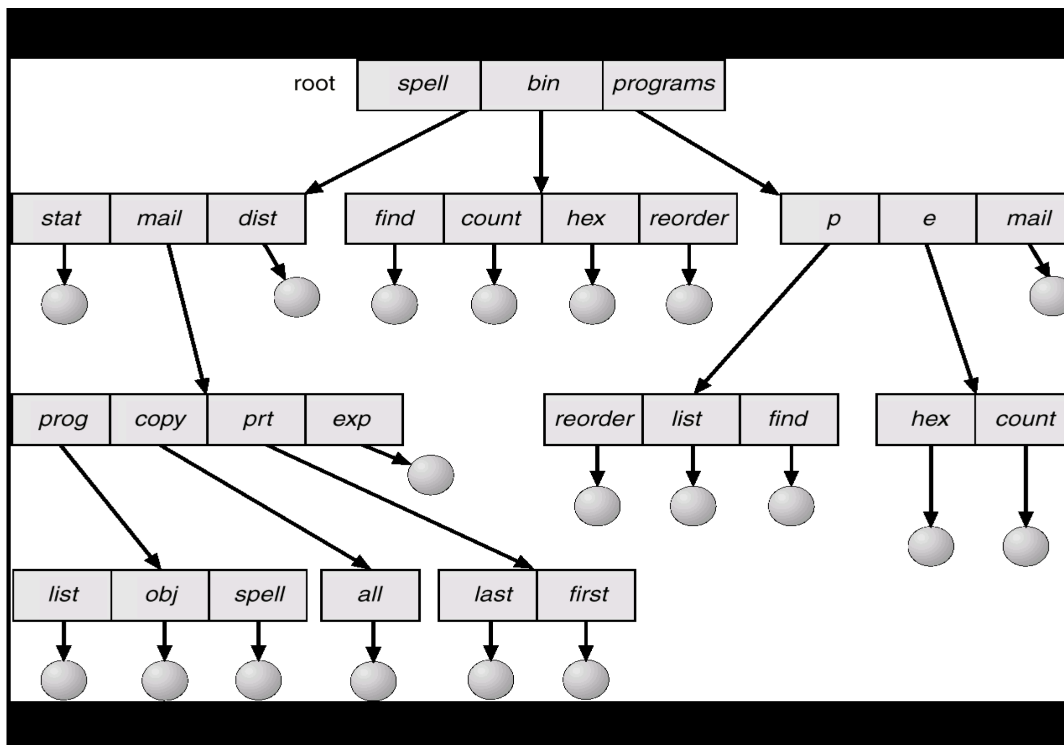
Thường thì mỗi người dùng có thư mục hiện hành. Thư mục hiện hành chứa hầu hết các tập tin người dùng hiện đang quan tâm. Khi tham khảo được thực hiện tới tập tin, thư mục hiện hành được tìm. Nếu một tập tin được yêu cầu mà nó không có trong thư mục hiện hành thì người dùng phải xác định tên đường dẫn hay chuyển thư mục hiện hành tới thư mục quản lý tập tin đó. Để thay đổi thư mục, một lời gọi hệ thống được cung cấp kèm theo tên thư mục như là tham số và dùng nó để định nghĩa lại thư mục hiện hành. Do đó, người dùng có thể thay đổi thư mục hiện hành bất cứ khi nào người dùng muốn.

Thư mục hiện hành khởi đầu của người dùng được gán khi công việc người dùng bắt đầu hay người dùng đăng nhập vào hệ thống. Hệ điều hành tìm tập tin tính toán để xác định mục từ cho người dùng này. Trong tập tin tính toán là con trỏ chỉ tới thư mục khởi đầu của người dùng. Con trỏ này được chép tới một biến cục bộ cho người dùng xác định thư mục hiện hành khởi đầu.

Tên đường dẫn có hai kiểu: tên đường dẫn tuyệt đối và tên đường dẫn tương đối. Một đường dẫn tuyệt đối bắt đầu từ gốc và theo sau là đường dẫn xuống tới tập tin xác định, cho tên các thư mục trên đường dẫn. Tên đường dẫn tương đối định nghĩa một đường dẫn từ thư mục hiện hành. Thí dụ, trong hệ thống tập tin có cấu trúc cây như hình IX-7,

nếu thư mục hiện hành là root/spell/mail thì tên đường dẫn tương đối prt/first tham chiếu tới cùng tập tin nhưng tên đường dẫn tuyệt đối root/spell/mail/prt/first.

Quyết định một chính sách trong cấu trúc thư mục cây là cách để quản lý việc xoá một thư mục. Nếu một thư mục rỗng, mục từ của nó trong thư mục chứa bị xoá. Tuy nhiên, giả sử thư mục bị xoá không rỗng, nhưng chứa nhiều tập tin và thư mục con; một trong hai tiếp cận có thể được thực hiện. Một số hệ thống như MS-DOS sẽ không xoá một thư mục nếu nó không rỗng. Do đó, để xoá một thư mục, người dùng trước hết phải xoá tất cả tập tin trong thư mục đó. Nếu bất cứ thư mục con tồn tại, thủ tục này phải được áp dụng đệ qui tới chúng để mà chúng có thể bị xoá. Tiếp cận này dẫn đến lượng công việc lớn.



Hình IX-7 cấu trúc thư mục dạng cây

Một tiếp cận khác được thực hiện bởi lệnh rm của UNIX cung cấp tuỳ chọn mà khi một yêu cầu được thực hiện để xoá một thư mục, tất cả tập

tin và thư mục con của thư mục đó cũng bị xoá. Tiếp cận này tương đối đơn giản để cài đặt; chọn lựa này là một chính sách. Chính sách sau đó tiện dụng hơn nhưng nguy hiểm hơn vì toàn bộ cấu trúc thư mục có thể bị xoá với một lệnh. Nếu lệnh đó được cấp phát bị lỗi, một số lượng lớn tập tin và thư mục cần được phục hồi từ các băng từ sao lưu.

Với một hệ thống thư mục cấu trúc cây, người dùng có thể truy xuất tới các tập tin của họ và các tập tin của người dùng khác. Thí dụ, người dùng B có thể truy xuất các tập tin của người dùng A bằng cách xác định tên đường dẫn của chúng. Người dùng B có thể xác định tên đường dẫn tương đối hay tuyệt đối. Người dùng B có thể chuyển thư mục hiện hành tới thư mục của người dùng A và truy xuất các tập tin bằng tên của chúng. Một số hệ thống cũng cho phép người dùng định nghĩa đường dẫn tìm kiếm của chính họ. Trong trường hợp này, người dùng B có thể định nghĩa đường dẫn tìm kiếm của mình là (1) thư mục cục bộ của mình, (2) thư mục tập tin hệ thống và (3) thư mục của người dùng A, theo thứ tự đó. Tập tin có thể được tham khảo đơn giản bằng tên với điều kiện tên tập tin của người dùng A không xung đột với tên của một tập tin cục bộ hay tập tin hệ thống.

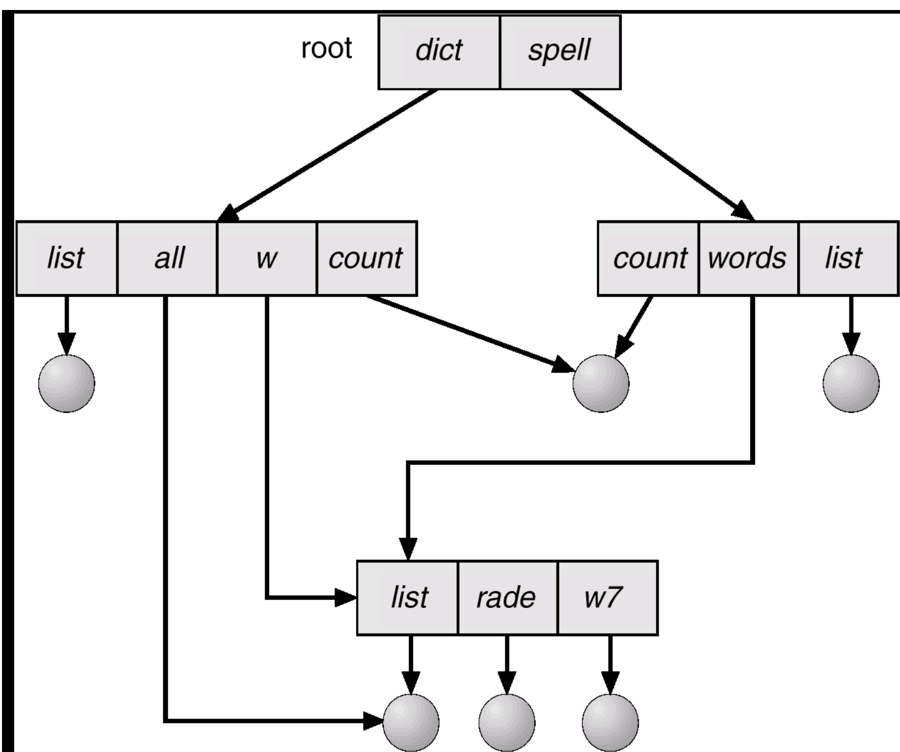
Cấu trúc thư mục dạng đồ thị không chứa chu trình

Xét hai người lập trình đang làm việc trên một dự án chung. Các tập tin gắn với dự án đó có thể được lưu trong thư mục con, tách rời chúng từ các dự án khác và các tập tin của hai người lập trình. Nhưng vì cả hai người lập trình có trách nhiệm ngang nhau trong dự án, cả hai muốn thư mục con ở trong các thư mục của chính họ. Thư mục con nên được chia sẻ. Một thư mục hay tập tin sẽ tồn tại trong hệ thống tập tin trong hai (hay nhiều hơn) nơi tại một thời điểm.

Cấu trúc cây ngăn cản việc chia sẻ các tập tin và thư mục. Một đồ thị không chứa chu trình (acyclic graph) cho phép thư mục chia sẻ thư mục con và tập tin (hình IX-8). Cùng tập tin và thư mục con có thể ở trong hai thư mục khác nhau. Một đồ thị không chứa chu trình là trường hợp tổng quát của cơ chế thư mục có cấu trúc cây.

Một tập tin (hay thư mục) được chia sẻ không giống như hai bản sao của một tập tin. Với hai bản sao, mỗi người lập trình có thể thích hiển thị bản sao hơn bản gốc, nhưng nếu một người lập trình thay đổi nội dung tập tin, những thay đổi sẽ không xuất hiện trong bản sao của người còn lại. Với một tập tin được chia sẻ, chỉ một tập tin thực sự tồn tại vì thế bất cứ sự thay đổi được thực hiện bởi một người này lập tức nhìn thấy bởi người dùng khác. Việc chia sẻ là rất quan trọng cho các thư mục con; một tập tin mới được tạo bởi người này sẽ tự động xuất hiện trong tất cả thư mục con được chia sẻ.

Khi nhiều người đang làm việc như một nhóm, tất cả tập tin họ muốn chia sẻ có thể đặt vào một thư mục. Các UFD của tất cả thành viên trong nhóm chứa thư mục của tập tin được chia sẻ như một thư mục con. Ngay cả khi có một người dùng, tổ chức tập tin của người dùng này yêu cầu rằng một số tập tin được đặt vào các thư mục con khác nhau. Thí dụ, một chương trình được viết cho một dự án nên đặt trong thư mục của tất cả chương trình và trong thư mục cho dự án đó.



Hình IX-8 cấu trúc đồ thị không chứa chu trình

Các tập tin và thư mục con được chia sẻ có thể được cài đặt trong nhiều cách. Cách thông dụng nhất được UNIX dùng là tạo một mục từ thư mục được gọi là liên kết. Một liên kết là một con trỏ chỉ tới một tập tin hay thư mục con khác. Thí dụ, một liên kết có thể được cài đặt như tên đường dẫn tuyệt đối hay tương đối. Khi một tham chiếu tới tập tin được thực hiện, chúng ta tìm kiếm thư mục. Nếu mục từ thư mục được đánh dấu như một liên kết thì tên tập tin thật sự (hay thư mục) được cho. Chúng ta phân giải liên kết bằng cách sử dụng tên đường dẫn để định vị tập tin thật sự. Những liên kết được xác định dễ dàng bởi định dạng trong mục từ thư mục và được định rõ bằng các con trỏ gián tiếp. Hệ điều hành bỏ qua các liên kết này khi duyệt qua cây thư mục để lưu giữ cấu trúc không chứa chu trình của hệ thống.

Một tiếp cận khác để cài đặt các tập tin được chia sẻ là nhân bản tất cả thông tin về chúng trong cả hai thư mục chia sẻ. Do đó, cả hai mục từ là giống hệt nhau. Một liên kết rất khác từ mục từ thư mục gốc. Tuy nhiên, nhân bản mục từ thư mục làm cho bản gốc và bản sao không khác nhau. Một vấn đề chính với nhân bản mục từ thư mục là duy trì tính không đổi nếu tập tin bị sửa đổi.

Một cấu trúc thư mục đồ thị không chứa chu trình linh hoạt hơn cấu trúc cây đơn giản nhưng nó cũng phức tạp hơn. Một số vấn đề phải được xem xét cẩn thận. Một tập tin có nhiều tên đường dẫn tuyệt đối. Do đó, các tên tập tin khác nhau có thể tham chiếu tới cùng một tập tin. Trường hợp này là tương tự như vấn đề bí danh cho các ngôn ngữ lập trình. Nếu chúng ta đang cố gắng duyệt toàn bộ hệ thống tập tin-để tìm một tập tin, để tập hợp các thông tin thống kê trên tất cả tập tin, hay chép tất cả tập tin tới thiết bị lưu dự phòng-vấn đề này trở nên lớn vì chúng ta không muốn duyệt các cấu được chia sẻ nhiều hơn một lần.

Một vấn đề khác liên quan đến việc xoá. Không gian được cấp phát tới tập tin được chia sẻ bị thu hồi và sử dụng lại khi nào? một khả năng là xoá bỏ tập tin bất cứ khi nào người dùng xoá nó, nhưng hoạt động này để lại con trỏ chỉ tới một tập tin không tồn tại. Trong trường hợp xấu

hơn, nếu các con trỏ tập tin còn lại chứa địa chỉ đĩa thật sự và không gian được dùng lại sau đó cho các tập tin khác, các con trỏ này có thể chỉ vào phần giữa của tập tin khác.

Trong một hệ thống mà việc chia sẻ được cài đặt bởi liên kết biểu tượng, trường hợp này dễ dàng quản lý hơn. Việc xoá một liên kết không cần tác động tập tin nguồn, chỉ liên kết bị xoá. Nếu chính tập tin bị xoá, không gian cho tập tin này được thu hồi, để lại các liên kết chơi vơi. Chúng ta có thể tìm các liên kết này và xoá chúng, nhưng nếu không có danh sách các liên kết được nối kết, việc tìm kiếm này sẽ tốn rất nhiều chi phí. Một cách khác, chúng ta có thể để lại các liên kết này cho đến khi nó được truy xuất. Tại thời điểm đó, chúng ta xác định rằng tập tin của tên được cho bởi liên kết không tồn tại và có thể bị lỗi để phục hồi tên liên kết; truy xuất này được đối xử như bất cứ tên tập tin không hợp lệ khác. Trong trường hợp UNIX, các liên kết biểu tượng được để lại khi một tập tin bị xoá và nó cho người dùng nhận thấy rằng tập tin nguồn đã mất hay bị thay thế. Microsoft Windows (tất cả ấn bản) dùng cùng tiếp cận.

Một tiếp cận khác đối với việc xoá là giữ lại tập tin cho tới khi tất cả tham chiếu tới nó bị xoá. Để cài đặt tiếp cận này, chúng ta phải có một số cơ chế để xác định rằng tham chiếu cuối cùng tới tập tin bị xoá. Chúng ta giữ danh sách của tất cả tham chiếu tới một tập tin (các mục từ thư mục hay các liên kết biểu tượng). Khi một liên kết hay bản sao của mục từ thư mục được thiết lập, một mục từ mới được thêm tới danh sách tham chiếu tập tin. Khi một mục từ thư mục hay liên kết bị xoá, chúng ta gỡ bỏ mục từ của nó trên danh sách. Tập tin này bị xoá khi danh sách tham chiếu tập tin của nó là rỗng.

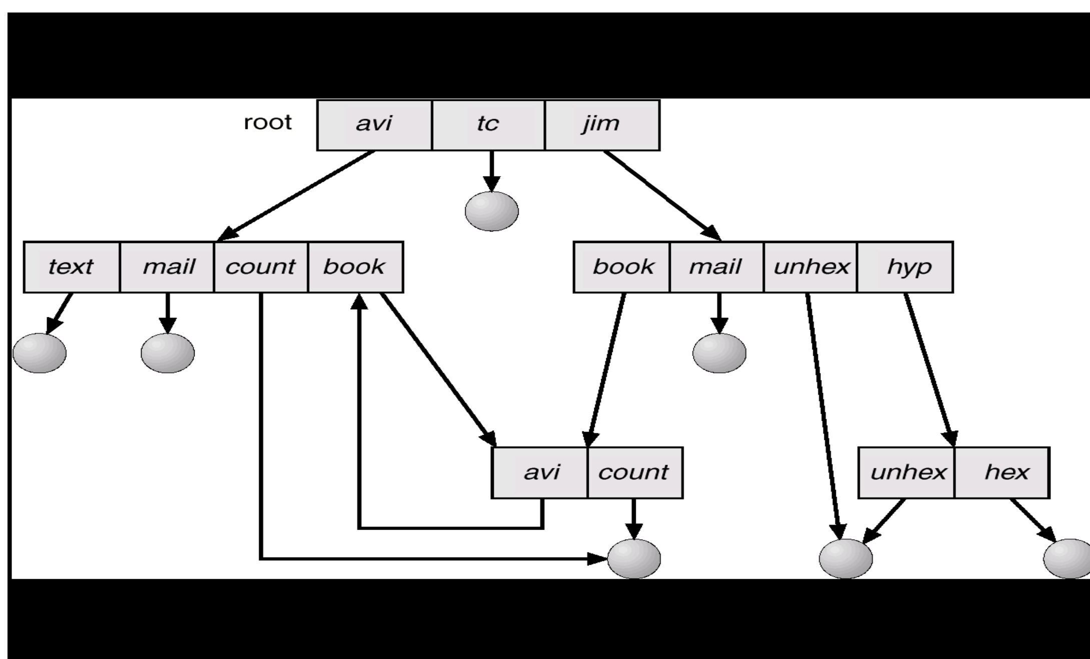
Trở ngại với tiếp cận này là kích thước của danh sách tham chiếu thay đổi và có thể rất lớn. Tuy nhiên, chúng ta thật sự không cần giữ toàn bộ danh sách-chúng ta chỉ cần giữ số đếm của số tham chiếu. Một liên kết mới hay mục từ thư mục mới sẽ tăng số đếm tham chiếu; xoá một liên kết hay mục từ sẽ giảm số đếm. Khi số đếm là 0, tập tin có thể được xoá; không còn tham chiếu nào tới nó. Hệ điều hành UNIX dùng tiếp cận này cho các liên kết không biểu tượng (hay liên kết cứng), giữ một số

đếm tham chiếu trong khối thông tin tập tin (hay inode). Bằng cách ngăn cản hiệu quả nhiều tham chiếu tới các thư mục, chúng ta duy trì cấu trúc đồ thị không chứa chu trình.

Để tránh vấn đề này, một số hệ thống không cho phép thư mục hay liên kết được chia sẻ. Thí dụ, trong MS-DOS, cấu trúc thư mục là một cấu trúc cây hơn là đồ thị không chứa chu trình.

Cấu trúc thư mục dạng đồ thị tổng quát

Một vấn đề lớn trong việc dùng cấu trúc đồ thị không chứa chu trình là đảm bảo rằng không có chu trình trong đồ thị. Nếu chúng ta bắt đầu với thư mục hai cấp và cho phép người dùng tạo thư mục con, một thư mục có cấu trúc cây tạo ra. Dễ thấy rằng thêm các tập tin và thư mục con mới tới một thư mục có cấu trúc cây đã có vẫn bảo đảm tính tự nhiên của cấu trúc cây. Tuy nhiên, khi chúng ta liên kết một thư mục cấu trúc cây đã có, cấu trúc cây bị phá vỡ hình thành một đồ thị đơn giản (như hình IX-9).



Hình IX-9 thư mục đồ thị tổng quát

Một lợi điểm chính của đồ thị không chứa chu trình là tương đối đơn giản trong giải thuật duyệt đồ thị và xác định khi không có tham chiếu nữa tới tập tin. Chúng ta muốn tránh duyệt các phần được chia sẻ của đồ thị không chứa chu trình hai lần để tăng năng lực. Nếu chúng ta chỉ tìm một thư mục con được chia sẻ cho một tập tin xác định, chúng ta muốn tránh việc tìm kiếm thư mục con đó một lần nữa; tìm kiếm lần hai sẽ lãng phí thời gian.

Nếu các chu trình được cho phép tồn tại trong thư mục, chúng ta muốn tránh tìm kiếm bất cứ thành phần nào hai lần vì tính đúng đắn cũng như năng lực. Một giải thuật được thiết kế nghèo nàn dẫn tới vòng lặp vô tận trên các chu trình. Một giải pháp là giới hạn số lượng thư mục sẽ được truy xuất trong quá trình tìm kiếm.

Một vấn đề tương tự tồn tại khi chúng ta cố gắng xác định khi nào một tập tin có thể bị xoá. Như với cấu trúc thư mục đồ thị không chứa chu trình, một giá trị 0 trong số đếm tham chiếu có nghĩa là không còn tham chiếu nữa tới tập tin hay thư mục và tập tin có thể bị xoá. Tuy nhiên, khi có chu trình tồn tại, số đếm tham chiếu khác 0 ngay cả khi không còn tham chiếu tới thư mục hay tập tin. Sai sót này dẫn tới khả năng tham chiếu chính nó (hay chu trình) trong cấu trúc thư mục. Trong trường hợp này, chúng ta cần dùng cơ chế thu dọn rác (garbage colection) để xác định khi tham chiếu cuối cùng bị xoá và không gian đĩa có thể được cấp phát lại. Thu dọn rác liên quan đến việc duyệt toàn bộ hệ thống tập tin, đánh dấu mọi thứ có thể được truy xuất. Sau đó, duyệt lần hai tập hợp mọi thứ không được đánh dấu trên danh sách không gian trống. Tuy nhiên, thu dọn rác cho một đĩa dựa trên hệ thống tập tin rất mất thời gian và do đó hiếm khi được thực hiện.

Thu dọn rác cần thiết chỉ vì có chu trình trong đồ thị. Do đó, cấu trúc đồ thị không chứa chu trình dễ hơn nhiều. Khó khăn là tránh chu trình khi các liên kết mới được thêm vào cấu trúc. Chúng ta biết như thế nào và khi nào một liên kết mới sẽ hình thành chu trình? Có nhiều giải thuật phát hiện các chu trình trong đồ thị; tuy nhiên chi phí tính toán cao, đặc biệt khi

đồ thị ở trên đĩa lưu trữ. Một giải thuật đơn giản hơn trong trường hợp đặc biệt của thư mục và liên kết là bỏ qua liên kết khi duyệt qua thư mục. Chu trình có thể tránh được và không có chi phí thêm xảy ra.

Gắn hệ thống tập tin

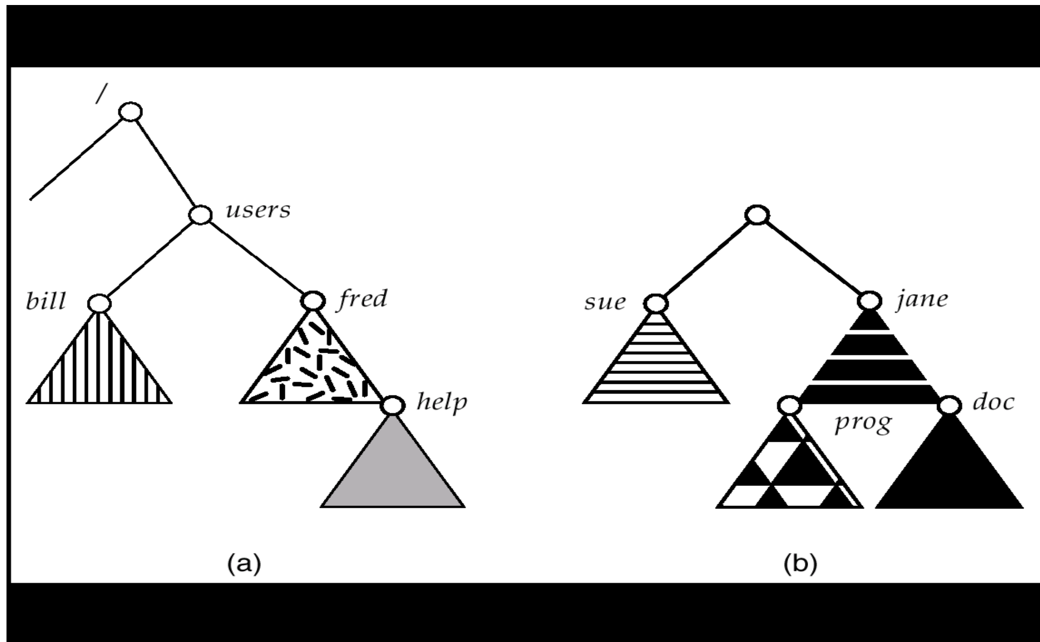
Giống như một tập tin phải được mở trước khi nó được dùng, một hệ thống tập tin phải được gắn vào (mounted) trước khi nó có thể sẵn dùng cho các quá trình trên hệ thống. Đặc biệt hơn, cấu trúc thư mục có thể được xây dựng trên nhiều phân khu, mà phải được gắn vào để làm cho chúng sẵn dùng trong không gian tên hệ thống tập tin.

Thủ tục gắn vào không phức tạp. Hệ điều hành được cho tên của thiết bị và vị trí trong cấu trúc tập tin tại nơi nó được gán vào hệ thống tập tin (điểm gắn-mount point). Điển hình, một điểm gắn thường là thư mục rỗng nơi hệ thống tập tin sẽ được gắn vào. Thí dụ, trên hệ thống UNIX, một hệ thống tập tin chứa các thư mục dành riêng cho người dùng (home directory) có thể được gắn vào như /home; sau đó để truy xuất tới cấu trúc tập tin trong hệ thống tập tin đó, người dùng có thể đến trước các tên thư mục /home, như trong /home/jane. Gắn vào hệ thống tập tin đó dưới /users sẽ dẫn tới tên đường dẫn /users/jane để đạt cùng thư mục.

Tiếp theo, hệ điều hành kiểm tra thiết bị chứa một hệ thống tập tin hợp lệ không. Nó thực hiện như thế bằng cách yêu cầu trình điều khiển thiết bị đọc thư mục thiết bị và kiểm tra thư mục có định dạng như mong muốn. Cuối cùng, hệ điều hành ghi nhận trong cấu trúc thư mục của nó rằng hệ thống tập tin được gắn vào tại điểm gắn xác định. Cơ chế này cho phép hệ điều hành duyệt cấu trúc thư mục của nó, chuyển qua lại giữa các hệ thống tập tin một cách hợp lý.

Để hiển thị việc gắn tập tin, xem xét hệ thống tập tin được mô tả như hình IX-10, ở đây hình này hình tam giác hiển thị cây con của các thư mục đang quan tâm. Trong hình (a) hiển thị hệ thống tập tin đã có, trong hình (b) hiển thị một phân khu chưa được gắn vào /device/dsk. Tại điểm này, chỉ các tập tin trên hệ thống tập tin đã tồn tại mới có thể được truy xuất. Trong hình IX-11, hiển thị các ảnh hưởng của việc gắn vào phân

khu này trên /device/dsk qua /users. Nếu phân khu này chưa được gắn, hệ thống tập tin được phục hồi tới trường hợp được mô tả như hình IX-10.

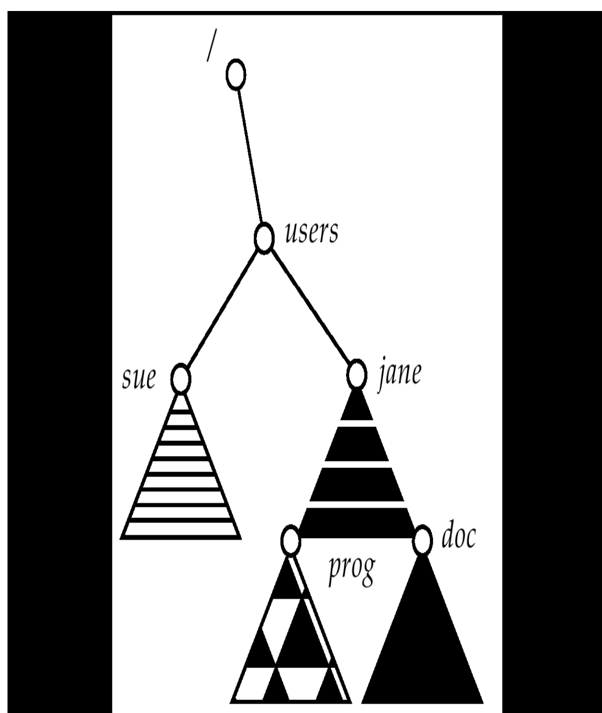


Hình IX-10 Hệ thống tập tin. (a) đã tồn tại. (b) phân khu chưa được gắn

Hệ thống này áp đặt các ngữ nghĩa để phân cấp chức năng. Thí dụ, một hệ thống không được phép vượt qua một thư mục chứa các tập tin hay làm hệ thống tập tin được gắn vào sẵn dùng tại thư mục đó và che đậy các tập tin đã có của thư mục cho đến khi hệ thống tập tin được gỡ ra (unmounted), kết thúc việc sử dụng hệ thống tập tin và cho phép truy xuất tới các tập tin trong thư mục đó.

Xét các hoạt động của hệ điều hành Macintosh. Bất cứ khi nào hệ thống gặp một đĩa tại thời điểm đầu tiên (đĩa cứng tại thời điểm khởi động hay đĩa mềm tại thời điểm chèn đĩa vào ổ), hệ điều hành Macintosh tìm kiếm một hệ thống tập tin cho thiết bị đó. Nếu nó tìm thấy một hệ thống tập tin, nó tự động gắn hệ thống tập tin tại mức gốc, thêm biểu tượng thư mục trên màn hình được ghi nhãn với tên của hệ thống tập tin (như được lưu trong thư mục thiết bị). Sau đó, người dùng có thể nhấp vào biểu tượng để hiển thị hệ thống tập tin vừa được gắn vào.

Họ hệ điều hành Microsoft Windows (95, 98, NT, 2000) duy trì một cấu trúc thư mục hai cấp mở rộng với các thiết bị và phân khu được gán một ký tự ổ đĩa. Các phân khu có cấu trúc thư mục đồ thị tổng quát được gắn liền với ký tự ổ đĩa. Sau đó, đường dẫn tới một tập tin xác định có dạng ký tự ổ đĩa:\đường dẫn\tới\tập tin. Những hệ điều hành này tự phát hiện tất cả thiết bị và gắn vào tất cả hệ thống tập tin được định vị tại thời điểm khởi động. Trong một vài hệ thống, như UNIX, các lệnh gắn vào là hiện (explicit). Một tập tin cấu hình hệ thống chứa danh sách thiết bị và các điểm gắn vào để tự động gắn vào tại thời điểm khởi động, nhưng những gắn vào khác phải được thực hiện thủ công.



Hình IX-11 Điểm gắn vào

Chia sẻ tập tin

Nhiều người dùng

Khi hệ điều hành cung cấp nhiều người dùng, các vấn đề chia sẻ tập tin, đặt tên tập tin, và bảo vệ tập tin trở nên quan trọng. Đối với một cấu trúc thư mục cho phép các tập tin được chia sẻ bởi nhiều người dùng, hệ thống phải dàn xếp việc chia sẻ tập tin. Mặc định, hệ thống có thể cho phép một người dùng truy xuất các tập tin của người dùng khác hay nó yêu cầu rằng một người dùng gán quyền truy xuất cụ thể tới các tập tin.

Để cài đặt chia sẻ và bảo vệ, hệ thống phải duy trì nhiều thuộc tính tập tin và thư mục hơn trên hệ thống đơn người dùng. Mặc dù, có nhiều tiếp cận cho chủ đề này, hầu hết các hệ thống đưa ra khái niệm người sở hữu (owner) và nhóm (group) tập tin/thư mục. Người sở hữu là người dùng có thể thay đổi các thuộc tính, gán truy xuất, và có hầu hết điều khiển qua tập tin và thư mục. Thuộc tính nhóm của tập tin được dùng để định nghĩa tập hợp con các người dùng có thể chia sẻ truy xuất tới tập tin.

Hệ thống tập tin ở xa

Sự phát triển của mạng cho phép giao tiếp giữa các máy tính ở xa. Mạng cho phép chia sẻ các tài nguyên trải rộng trong một khu hay thậm chí khắp thế giới. Một tài nguyên quan trọng để chia sẻ là dữ liệu ở dạng tập tin. Thông quan sự phát triển mạng và công nghệ tập tin, phương pháp chia sẻ tập tin thay đổi. Trong phương pháp đầu tiên được cài đặt, người dùng truyền tập tin giữa các máy tính bằng chương trình gọi là ftp. Phương pháp quan trọng thứ hai là hệ thống tập tin phân tán (distributed file system-DFS) trong đó, các thư mục ở xa có thể được nhìn thấy từ máy cục bộ. Trong một số cách, phương pháp thứ ba, World Wide Web là một sự trở lại của phương pháp đầu tiên. Một trình duyệt được yêu cầu để đạt được truy xuất các tập tin từ xa và các thao tác riêng biệt được dùng để truyền tập tin.

Bảo vệ

Khi thông tin được giữ trong một hệ thống máy tính, chúng ta muốn giữ nó an toàn từ hỏng hóc vật lý (khả năng tin cậy) và những truy xuất không hợp lý (bảo vệ).

Khả năng tin cậy thường được cung cấp bởi nhân bản các tập tin. Nhiều máy tính có các chương trình hệ thống tự động chép các tập tin trên đĩa tới băng từ tại những khoảng thời gian đều đặn để duy trì một bản sao. Hệ thống tập tin có thể bị hỏng bởi phần cứng, thay đổi đột ngột về điện, nhiệt độ tăng cao,..các tập tin có thể bị xoá do rủi ro. Những con bọ (bugs) trong phần mềm hệ thống tập tin có thể làm cho nội dung tập tin bị mất.

Bảo vệ có thể được cung cấp trong nhiều cách. Đối với một hệ thống người dùng đơn nhỏ, chúng ta có thể cung cấp sự bảo vệ bằng cách gỡ bỏ các đĩa mềm và khoá chúng trong ngăn kéo. Tuy nhiên, trong hệ thống đa người dùng, những cơ chế khác được yêu cầu.

Các kiểu truy xuất

Nhu cầu bảo vệ tập tin là một kết quả trực tiếp của khả năng để truy xuất tập tin. Hệ thống không cho phép truy xuất các tập tin của người dùng khác thì không cần bảo vệ. Do đó, chúng ta có thể cung cấp sự bảo vệ toàn diện bằng cách cấm truy xuất. Một cách khác, chúng ta có thể cung cấp truy xuất thoải mái và không cần bảo vệ. Cả hai tiếp cận là quá cực đoan cho các sử dụng thông thường. Yêu cầu truy xuất được kiểm soát là gì?

Các cơ chế bảo vệ cung cấp truy xuất được kiểm soát bằng cách giới hạn kiểu truy xuất tập tin có thể thực hiện. Truy xuất được phép hay bị từ chối phụ thuộc nhiều yếu tố, một trong những yếu tố là kiểu truy xuất được yêu cầu. Nhiều kiểu thao tác có thể được kiểm soát:

- Đọc (Read): đọc từ tập tin.
- Viết (Write): viết hay viết lại tập tin.
- Thực thi (Execute): nạp tập tin vào bộ nhớ và thực thi nó.

- Chèn cuối (Append): viết thông tin mới vào cuối tập tin.
- Xoá (Delete): xoá tập tin và giải phóng không gian để có thể dùng lại
- Liệt kê (List): liệt kê tên và thuộc tính của tập tin

Những thao tác khác như đổi tên, chép, soạn thảo tập tin có thể cũng được kiểm soát. Tuy nhiên, đối với nhiều hệ thống, các chức năng cao hơn này có thể được cài đặt bởi một chương trình hệ thống thực hiện lời gọi hệ thống cấp thấp hơn. Bảo vệ được cung cấp chỉ tại cấp thấp hơn. Thí dụ, chép một tập tin có thể được cài đơn giản bởi một chuỗi các yêu cầu đọc. Trong trường hợp này, người dùng với truy xuất đọc cũng có thể làm cho tập tin được chép, in,..

Nhiều cơ chế bảo vệ được đề nghị. Mỗi cơ chế có lợi điểm và nhược điểm và phải phù hợp cho ứng dụng được dự định. Một hệ thống máy tính nhỏ được dùng chỉ bởi một vài thành viên của một nhóm nghiên cứu có thể không cần cùng kiểu bảo vệ như máy tính của công ty lớn.

Kiểm soát truy xuất

Tiếp cận thông dụng nhất đối với vấn đề bảo vệ là thực hiện truy xuất phụ thuộc định danh của người dùng. Những người dùng khác nhau cần các kiểu truy xuất khác nhau tới một tập tin hay thư mục. Cơ chế thông dụng nhất để cài đặt truy xuất phụ thuộc định danh là gắn với mỗi tập tin và thư mục một danh sách kiểm soát truy xuất (access-control list-ACL) xác định tên người dùng và kiểu truy xuất được phép cho mỗi người dùng. Khi một người dùng yêu cầu truy xuất tới một tập tin cụ thể, hệ điều hành kiểm tra danh sách truy xuất được gắn tới tập tin đó. Nếu người dùng đó được liệt kê cho truy xuất được yêu cầu, truy xuất được phép. Ngược lại, sự vi phạm bảo vệ xảy ra và công việc của người dùng đó bị từ chối truy xuất tới tập tin.

Tiếp cận này có lợi điểm của việc cho phép các phương pháp truy xuất phức tạp. Vấn đề chính với các danh sách truy xuất là chiều dài của nó. Nếu chúng ta muốn cho phép mọi người dùng đọc một tập tin, chúng ta

phải liệt kê tất cả người dùng với truy xuất đọc. Kỹ thuật này có hai kết quả không mong muốn:

- Xây dựng một danh sách như thế có thể là một tác vụ dài dòng và không đáng, đặc biệt nếu chúng ta không biết trước danh sách người dùng trong hệ thống.
- Mục từ thư mục, trước đó có kích thước cố định, bây giờ có kích thước thay đổi, dẫn đến việc quản lý không gian phức tạp hơn

Những vấn đề này có thể được giải quyết bởi việc dùng ấn bản cô đọng của danh sách truy xuất.

Để cô đọng chiều dài của danh sách kiểm soát truy xuất, nhiều hệ thống nhận thấy 3 sự phân cấp người dùng trong nối kết với mỗi tập tin:

- Người sở hữu (Owner): người dùng tạo ra tập tin đó
- Nhóm (Group): tập hợp người dùng đang chia sẻ tập tin và cần truy xuất tương tự là nhóm hay nhóm làm việc
- Người dùng khác (universe): tất cả người dùng còn lại trong hệ thống

Tiếp cận phổ biến gần đây nhất là kết hợp các danh sách kiểm soát truy xuất với người sở hữu, nhóm và cơ chế kiểm soát truy xuất được mô tả ở trên

Để cơ chế này làm việc hợp lý, các quyền và danh sách truy xuất phải được kiểm soát chặt chẽ. Kiểm soát này có thể đạt được trong nhiều cách. Thí dụ, trong hệ thống UNIX, các nhóm có thể được tạo và sửa đổi chỉ bởi người quản lý của tiện ích. Do đó, kiểm soát này đạt được thông qua giao tiếp người dùng.

Với việc phân cấp bảo vệ được giới hạn hơn, chỉ có ba trường được yêu cầu để xác định bảo vệ. Mỗi trường thường là một tập hợp các bit, mỗi trường cho phép hay ngăn chặn truy xuất được gắn với nó. Thí dụ, hệ thống UNIX định nghĩa 3 trường 3 bit-rwx, ở đây r kiểm soát truy xuất đọc, w kiểm soát truy xuất viết, x kiểm soát truy xuất thực thi. Một trường riêng rẻ được giữ cho người sở hữu, cho nhóm tập tin và cho tất

cả người dùng khác. Trong cơ chế này, 9 bits trên tập tin được yêu cầu để ghi lại thông tin bảo vệ.

Các tiếp cận bảo vệ khác

Một tiếp cận khác cho vấn đề bảo vệ là gắn mật khẩu với mỗi tập tin. Giống như truy xuất tới hệ thống máy tính thường được kiểm soát bởi một mật khẩu, truy xuất tới mỗi tập tin có thể được kiểm soát bởi một mật khẩu. Nếu các mật khẩu được chọn một cách ngẫu nhiên và thường được thay đổi thì cơ chế này có thể hiệu quả trong truy xuất có giới hạn tới tập tin cho những người dùng biết mật khẩu. Tuy nhiên, cơ chế này có nhiều nhược điểm. Thứ nhất, số lượng mật khẩu mà người dùng cần nhớ quá nhiều, làm cho cơ chế này không thực tế. Thứ hai, nếu chỉ một mật khẩu được dùng cho tất cả tập tin thì một khi nó bị phát hiện tất cả tập tin có thể truy xuất. Một số hệ thống cho phép người dùng gắn một mật khẩu tới một thư mục con hơn là với từng tập tin riêng rẻ để giải quyết vấn đề này. Thứ ba, thường chỉ một mật khẩu gắn với tất cả tập tin người dùng. Do đó, bảo vệ dựa trên cơ sở tất cả hay không có gì (all-or-nothing). Để cung cấp sự bảo vệ trên cấp độ chi tiết hơn chúng ta phải dùng nhiều mật khẩu.

Tóm tắt

Một tập tin là một kiểu dữ liệu trừu tượng được định nghĩa và được cài đặt bởi hệ điều hành. Nó là một chuỗi các mẫu tin luận lý. Một mẫu tin luận lý có thể là một byte, một dòng (có chiều dài cố định hay thay đổi), hay có thành phần dữ liệu phức tạp hơn. Hệ điều hành có thể hỗ trợ nhiều kiểu mẫu tin khác nhau hay để sự hỗ trợ đó tới một chương trình ứng dụng.

Mỗi thiết bị trong một tập tin giữ một bảng volume nội dung hay thư mục thiết bị liệt kê vị trí các tập tin trên thiết bị. Ngoài ra, nó có ích để tạo các thư mục cho phép các tập tin được tổ chức trong thư mục đó. Một thư mục đơn cấp trong hệ thống đơn người dùng gây ra các vấn đề

đặt tên vì mỗi tập tin phải có tên duy nhất. Thư mục hai cấp giải quyết vấn đề này bằng cách tạo một thư mục riêng cho mỗi người dùng. Mỗi người dùng có thư mục riêng, chứa tập tin riêng. Thư mục liệt kê các tập tin bằng tên và chứa những thông tin như vị trí tập tin trên đĩa, chiều dài, kiểu, người sở hữu, thời gian tạo, thời điểm dùng gần nhất,…

Tổng quát hóa tính tự nhiên của thư mục hai cấp là thư mục có cấu trúc cây. Thư mục có cấu trúc cây cho phép một người dùng tạo thư mục con để tổ chức các tập tin. Cấu trúc thư mục đồ thị không chứa chu trình cho phép các thư mục con và tập tin được chia sẻ nhưng tìm kiếm và xóa phức tạp. Một cấu trúc đồ thị tổng quát linh động hơn trong việc chia sẻ tập tin và thư mục, nhưng yêu cầu thu dọn rác để phục hồi không gian đĩa không được dùng.

Đĩa được phân chia thành một hay nhiều phân khu, mỗi phân khu chứa một hệ thống tập tin. Hệ thống tập tin này được gán vào cấu trúc đặt tên của hệ thống để làm cho chúng sẵn dùng. Cơ chế đặt tên khác nhau bởi các hệ điều hành khác nhau. Một khi được gán vào, các tập tin trong phân khu là sẵn dùng. Các hệ thống tập tin có thể được gỡ ra (unmount) để vô hiệu hóa truy xuất hay để bảo trì.

Chia sẻ tập tin phụ thuộc vào ngữ nghĩa được cung cấp bởi hệ thống. Các tập tin có nhiều người đọc, viết hay bị giới hạn việc chia sẻ. Hệ thống tập tin phân tán cho phép máy khách hàng gắn các phân khu hay thư mục vào từ nhiều server. Với điều kiện chúng có thể truy xuất nhau qua mạng. Các hệ thống tập tin ở xa có những thách thức về khả năng tin cậy, năng lực và bảo mật. Hệ thống thông tin được phân tán duy trì người dùng, máy chủ, thông tin truy xuất như khách hàng và thông tin trạng thái chia sẻ servers để quản lý việc sử dụng và truy xuất.

Vì các tập tin là cơ chế lưu trữ thông tin quan trọng trong hầu hết các hệ thống máy tính nên bảo vệ tập tin là cần thiết. Truy xuất tới các tập tin được kiểm soát riêng cho mỗi loại truy xuất-đọc, viết, thực thi, chèn cuối, xóa, liệt kê thư mục,..Bảo vệ tập tin có thể được cung cấp bởi mật khẩu, bởi danh sách truy xuất hay bởi những kỹ thuật phức tạp.

Cài đặt hệ thống tập tin
Mục đích Sau khi học xong chương này, người học nắm được những kiến thức sau: - Hiểu việc lưu trữ các tập tin và truy xuất các tập tin trên các thiết bị lưu trữ phụ. - Hiểu các phương pháp để thiết lập việc sử dụng tập tin - Hiểu cách cấp phát không gian đĩa, phục hồi không gian trống, ghi vết vị trí dữ liệu

Giới thiệu

Trong chương trước chúng ta thấy rằng, hệ thống tập tin cung cấp cơ chế cho việc lưu trữ trực tuyến (on-line storage) và truy xuất tới nội dung tập tin, gồm dữ liệu và chương trình. Hệ thống tập tin định vị vĩnh viễn trên thiết bị lưu trữ phụ. Các thiết bị này được thiết kế để quản lý lượng lớn thông tin không thay đổi.

Chương này tập trung chủ yếu với những vấn đề xoay quanh việc lưu trữ tập tin và truy xuất trên các thiết bị lưu trữ phụ. Chúng ta khám phá các cách để xây dựng cấu trúc sử dụng tập tin, cấp phát không gian đĩa và phục hồi không gian trống để ghi lại vị trí dữ liệu và để giao tiếp với các phần khác của hệ điều hành tới thiết bị lưu trữ phụ. Các vấn đề về năng lực được xem xét thông qua chương này.

Cấu trúc hệ thống tập tin

Đĩa cung cấp số lượng thiết bị lưu trữ phụ mà trên đó hệ thống tập tin được duy trì. Có hai đặc điểm làm đĩa trở thành phương tiện tiện dụng cho việc lưu trữ nhiều tập tin:

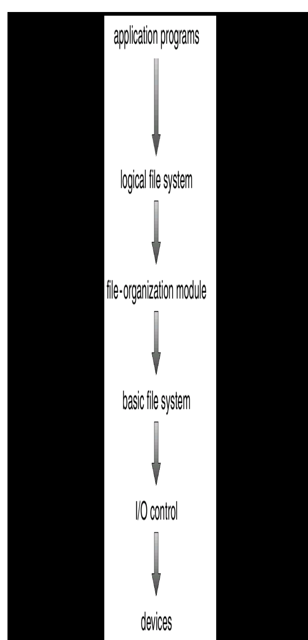
1. Chúng có thể được viết lại bằng cách thay thế; có thể đọc một khối từ đĩa, sửa một khối và viết nó ngược trở lại đĩa trong cùng vị trí.
2. Chúng có thể được truy xuất trực tiếp bất cứ khối thông tin nào trên đĩa.

Để cải tiến tính hiệu quả nhập/xuất, thay vì chuyển một byte tại một thời điểm, nhập/xuất chuyển giữa bộ nhớ và đĩa được thực hiện trong

đơn vị khối. Mỗi khối là một hay nhiều cung từ (sector). Phụ thuộc ổ đĩa, các cung từ biến đổi từ 32 bytes tới 4096 bytes; thường là 512 bytes.

Để cung cấp việc truy xuất hiệu quả và tiện dụng tới đĩa, hệ điều hành áp đặt một hay nhiều hệ thống tập tin để cho phép dữ liệu được lưu trữ, định vị và truy xuất lại dễ dàng. Một hệ thống tập tin đặt ra hai vấn đề thiết kế rất khác nhau. Vấn đề đầu tiên là định nghĩa hệ thống tập tin nên quan tâm đến người dùng như thế nào. Tác vụ này liên quan đến việc định nghĩa một tập tin và thuộc tính của nó, các thao tác được phép trên một tập tin và các giải thuật và cấu trúc cho việc tổ chức tập tin. Vấn đề thứ hai là tạo giải thuật và cấu trúc dữ liệu để ánh xạ hệ thống tập tin luận lý vào các thiết bị lưu trữ phụ.

Hệ thống tập tin thường được tạo thành từ nhiều cấp khác nhau. Cấu trúc được hiển thị trong hình X-1 là một thí dụ của thiết kế phân cấp. Mỗi cấp trong thiết kế dùng các đặc điểm của cấp thấp hơn để tạo các đặc điểm mới cho việc sử dụng bởi cấp cao hơn.



Hình X-1 hệ thống tập tin phân tầng

- Điều khiển nhập/xuất (I/O control): là cấp thấp nhất chứa các trình điều khiển thiết bị và các bộ quản lý ngắt để chuyển thông tin giữa bộ nhớ chính và hệ thống đĩa. Trình điều khiển thiết bị thường viết các mẫu bit xác định tới các vị trí trong bộ nhớ của bộ điều khiển nhập/xuất để báo với bộ điều khiển vị trí trên thiết bị nào và hoạt động gì xảy ra.
- Hệ thống tập tin cơ bản (basic file system) chỉ cần phát ra các lệnh thông thường tới các trình điều khiển thiết bị tương ứng để đọc và viết các khối vật lý trên đĩa. Mỗi khối vật lý được xác định bởi địa chỉ đĩa (thí dụ, đĩa 1, cyclinder 73, track 2, sector 10).
- Module tổ chức tập tin (file-organization module) biết các tập tin và các khối luận lý cũng như các khối vật lý. Bằng cách biết kiểu cấp phát tập tin được dùng và vị trí của tập tin, module tổ chức tập tin có thể dịch các địa chỉ khối luận lý thành các địa chỉ khối vật lý cho hệ thống tập tin cơ bản để truyền. Các khối luận lý của mỗi tập tin được đánh số từ 0 (hay 1) tới N, ngược lại các khối vật lý chứa dữ liệu thường không khớp với các số luận lý vì thế một thao tác dịch được yêu cầu để định vị mỗi khối. Module tổ chức tập tin cũng chứa bộ quản lý không gian trống (free-space manager), mà nó ghi vết các khối không được cấp phát và cung cấp các khối này tới module tổ chức tập tin khi được yêu cầu.
- Hệ thống tập tin luận lý (logical file system) quản lý thông tin siêu dữ liệu (metadata). Metadata chứa tất cả cấu trúc hệ thống tập tin, ngoại trừ dữ liệu thật sự (hay nội dung của các tập tin). Hệ thống tập tin luận lý quản lý cấu trúc thư mục để cung cấp module tổ chức tập tin những thông tin yêu cầu sau đó, được cho tên tập tin ký hiệu. Nó duy trì cấu trúc tập tin bằng khối điều khiển tập tin. Một khối điều khiển tập tin (file control block-FCB) chứa thông tin về tập tin, gồm người sở hữu, quyền và vị trí của nội dung tập tin.

Nhiều hệ thống tập tin được cài đặt hiện nay. Hầu hết hệ điều hành hỗ trợ nhiều hơn một hệ thống tập tin. Mỗi hệ điều hành có hệ thống tập tin dựa trên cơ sở đĩa. UNIX dùng hệ thống tập tin UNIX (UNIX file system-UFS) như là cơ sở. Windows NT hỗ trợ các định dạng tập tin FAT, FAT32 và NTFS cũng như CD-ROM, DVD và các định dạng hệ thống tập tin đĩa mềm. Bằng cách dùng cấu trúc phân cấp cho việc cài đặt hệ thống

tập tin, nên nhân bản mã là tối thiểu. Điều khiển nhập/xuất và mã hệ thống tập tin cơ bản có thể được dùng bởi nhiều hệ thống tập tin. Mỗi hệ thống tập tin có hệ thống tập tin luận lý và module tổ chức tập tin của chính nó.

Cài đặt hệ thống tập tin

Trong phần này chúng ta sẽ nghiên cứu các cấu trúc và các thao tác được dùng để cài đặt các thao tác của hệ thống tập tin.

Tổng quan

Nhiều cấu trúc trên đĩa và trên bộ nhớ được dùng để cài đặt một hệ thống tập tin. Các cấu trúc này thay đổi dựa trên hệ điều hành và hệ thống tập tin nhưng có một số nguyên tắc chung được áp dụng. Trên đĩa, hệ thống tập tin chứa thông tin về cách khởi động hệ điều hành được lưu trữ ở đó, tổng số khối, số và vị trí của các khối trống, cấu trúc thư mục, các tập tin riêng biệt.

Các cấu trúc trên đĩa gồm:

- Khối điều khiển khởi động (boot control block) có thể chứa thông tin được yêu cầu bởi hệ thống để khởi động một hệ điều hành từ phân khu đó. Nếu đĩa không chứa hệ điều hành thì khối này là rỗng. Điển hình, nó là khối đầu tiên của đĩa. Trong UFS, khối này được gọi là khối khởi động; trong NTFS, nó là cung khởi động phân khu (partition boot sector).
- Khối điều khiển phân khu (partition control block) chứa chi tiết về phân khu, như số lượng khối trong phân khu, kích thước khối, bộ đếm khối trống và con trỏ khối trống, bộ đếm FCB trống và con trỏ FCB. Trong UFS khối này được gọi là siêu khối (superblock); trong NTFS, nó là bảng tập tin chính (Master File Table)
- Một cấu trúc tập tin được dùng để tổ chức các tập tin

- Một FCB chứa nhiều chi tiết tập tin gồm các quyền tập tin, người sở hữu, kích thước, và vị trí của các khối dữ liệu. Trong UFS khối này được gọi là inode. Trong NTFS, thông tin này được lưu trong Master File Table dùng cấu trúc cơ sở dữ liệu quan hệ với một dòng cho một tập tin.

Thông tin trong bộ nhớ được dùng cho việc quản lý hệ thống tập tin và cải tiến năng lực qua lưu trữ (caching). Các cấu trúc này có thể bao gồm:

- Bảng phân khu trong bộ nhớ chứa thông tin về mỗi phân khu được gắn vào.
- Cấu trúc thư mục trong bộ nhớ quản lý thông tin thư mục của những thư mục vừa được truy xuất. (đối với các thư mục nơi mà các phân khu được gắn vào, nó có thể chứa một con trỏ chỉ tới bảng phân khu.)
- Bảng tập tin đang mở của hệ thống (system-wide open-file table) chứa bản sao của FCB của mỗi tập tin đang mở cũng như các thông tin khác.
- Bảng tập tin đang mở trên quá trình (per-process open-file table) chứa con trỏ chỉ tới mục từ tương ứng trong bảng tập tin đang mở của hệ thống cũng như những thông tin khác.

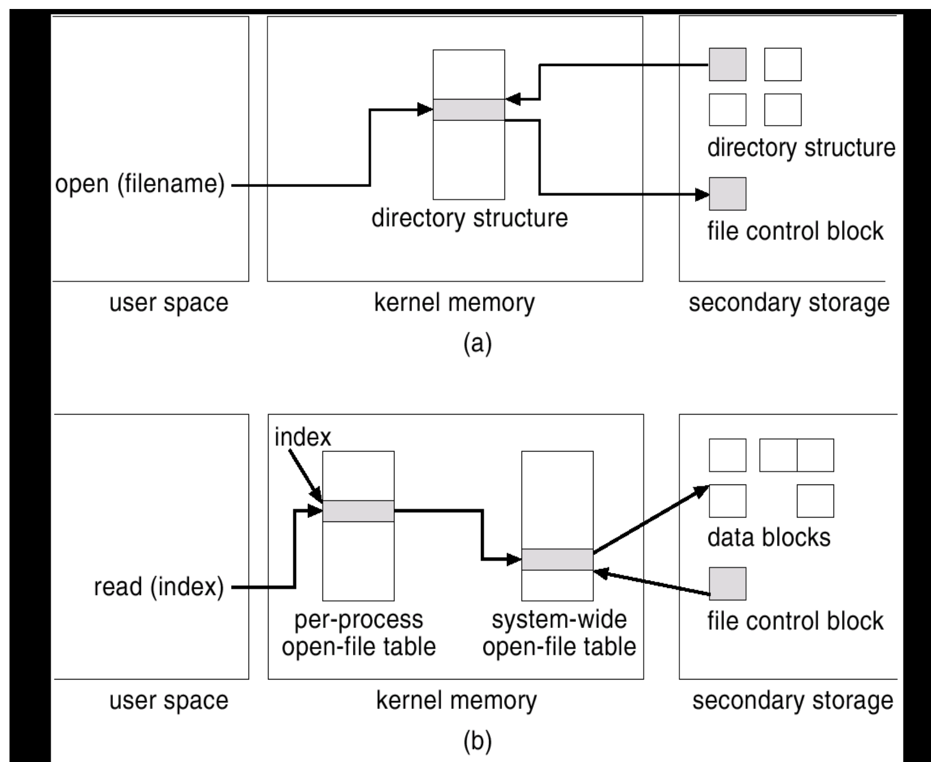
Để tạo một tập tin mới, một chương trình ứng dụng gọi hệ thống tập tin luận lý. Hệ thống tập tin luận lý biết định dạng của các cấu trúc thư mục. Để tạo một tập tin mới, nó cấp phát một FCB mới, đọc thư mục tương ứng vào bộ nhớ, cập nhật nó với tên tập tin mới và FCB, và viết nó trở lại đĩa. Một FCB điển hình được hiển thị trong hình X-2.

file permissions
file dates (create, access, write)
file owner, group, ACL
file size
file data blocks

Hình X-2 Một khối điều khiển tập tin điển hình

Một số hệ điều hành như UNIX xem một thư mục như là một tập tin-một tập tin với một trường kiểu hiển thị rằng nó là một thư mục. Các hệ điều hành khác như Windows NT cài đặt các lời gọi hệ thống riêng cho tập tin và thư mục và xem các thư mục như các thực thể tách rời từ các tập tin. Đối với cấu trúc lớn hơn, hệ thống tập tin luận lý có thể gọi module tổ chức tập tin để ánh xạ nhập/xuất thư mục vào số khối đĩa mà chúng được truyền trên cơ sở hệ thống tập tin và hệ thống điều khiển nhập/xuất. Module tổ chức tập tin cũng cấp phát các khối cho việc lưu trữ dữ liệu của tập tin.

Một tập tin được tạo, nó có thể được dùng cho nhập/xuất. Đầu tiên, nó phải được mở. Lời gọi open truyền tên tập tin tới hệ thống tập tin. Khi một tập tin được mở, cấu trúc thư mục thường được lưu vào bộ nhớ để tăng tốc độ các thao tác thư mục. Một khi tập tin được tìm thấy, FCB được chép vào bảng tập tin đang mở của hệ thống trong bộ nhớ. Bảng này không chỉ chứa FCB mà còn có các mục từ cho số đếm của số quá trình có mở tập tin.



Hình X-3 Cấu trúc hệ thống trong bộ nhớ. (a) mở tập tin. (b) đọc tập tin.

Tiếp theo, một mục từ được tạo trong bảng tập tin đang mở trên quá trình, với một con trỏ chỉ tới mục từ trong bảng hệ thống tập tin đang mở của hệ thống và một số trường khác. Các trường khác này có thể chứa con trỏ chỉ tới vị trí hiện hành trong tập tin (cho các thao tác read hay write tiếp theo) và chế độ truy xuất trong tập tin được mở. Lời gọi open trả về một con trỏ chỉ tới mục từ tương ứng trong bảng hệ thống tập tin trên quá trình. Sau đó, tất cả thao tác tập tin được thực hiện bằng con trỏ này. Tên tập tin không phải là một phần của bảng tập tin đang mở, hệ thống không dùng nó một khi FCB tương ứng được định vị trên đĩa. Tên được cho đối với mục từ rất đa dạng. Các hệ thống UNIX chỉ tới nó như một bộ mô tả tập tin (file discriptor); Windows 2000 chỉ tới nó như một bộ quản lý tập tin (file handle). Do đó, với điều kiện là tập tin không đóng, tất cả các thao tác tập tin được thực hiện trên bảng tập tin đang mở.

Khi một quá trình đóng tập tin, mục từ trong bảng trên quá trình bị xoá và bộ đếm số lần mở của mục từ hệ thống giảm. Khi tất cả người dùng

đóng tập tin, thông tin tập tin được cập nhật sẽ được chép trở lại tới cấu trúc thư mục dựa trên đĩa và mục từ bảng tập tin đang mở bị xóa.

Trong thực tế, lời gọi hệ thống open đầu tiên tìm bảng tập tin đang mở hệ thống để thấy nếu tập tin được sử dụng rồi bởi một quá trình khác. Nếu nó là mục từ bảng tập tin đang mở trên quá trình được tạo để chỉ tới bảng tập tin đang mở hệ thống đã có. Giải thuật này có thể tiết kiệm chi phí khi các tập tin đã mở rồi.

Các cấu trúc điều hành của việc cài đặt hệ thống tập tin được tóm tắt như hình X-3.

Hệ thống tập tin ảo

Một phương pháp nổi bật cho việc cài đặt nhiều loại hệ thống tập tin là viết các chương trình con thư mục và tập tin cho mỗi loại. Đúng hơn là hầu hết các hệ điều hành, gồm UNIX, dùng các kỹ thuật hướng đối tượng để đơn giản hóa, tổ chức, và module hóa việc cài đặt. Sử dụng các phương pháp này cho phép nhiều loại hệ thống tập tin khác nhau được cài đặt trong cùng cấu trúc, gồm các hệ thống tập tin mạng như NFS. Người dùng có thể truy xuất các tập tin được chứa trong nhiều hệ thống tập tin trên đĩa cục bộ, hay ngay cả trên các hệ thống tập tin sẵn dùng qua mạng.

Các cấu trúc dữ liệu và thủ tục được dùng để cô lập chức năng lời gọi hệ thống cơ bản từ các chi tiết cài đặt. Do đó, cài đặt hệ thống tập tin chứa ba tầng chính; nó được mô tả dưới dạng lưu đồ trong hình X-4. Tầng đầu tiên là giao diện hệ thống tập tin dựa trên cơ sở lời gọi open, read, write, close và các bộ mô tả tập tin.

Tầng thứ hai được gọi là hệ thống tập tin ảo (Virtual File System-VFS); nó phục vụ hai chức năng quan trọng:

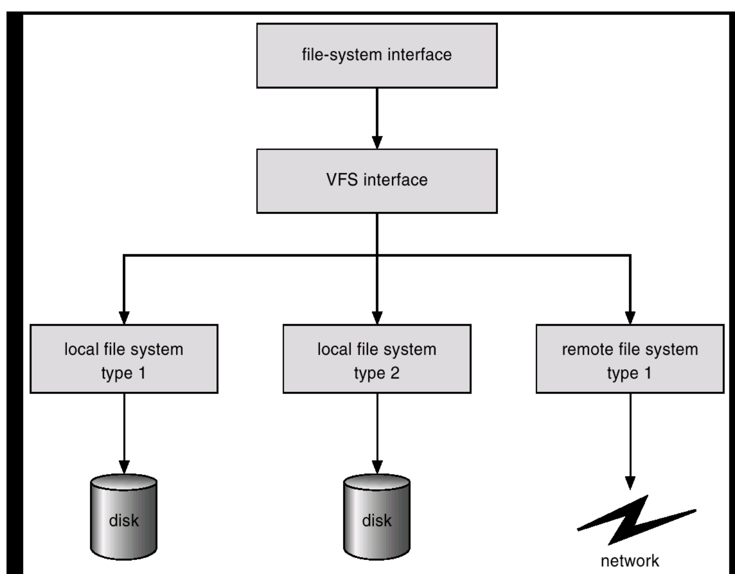
1. Nó tách biệt các thao tác hệ thống tập tin giống nhau từ việc cài đặt bằng cách định nghĩa một giao diện VFS rõ ràng. Nhiều cài đặt cho

giao diện VFS có thể cùng tồn tại trên cùng một máy, cho phép truy xuất trong suốt tới các loại hệ thống tập tin khác nhau được gắn cục bộ.

2. VFS dựa trên cấu trúc biểu diễn tập tin, được gọi là vnode, chứa một bộ gắn bằng số (numerical designator) cho tập tin duy nhất qua mạng. (Inode của UNIX là duy nhất chỉ trong một hệ thống tập tin đơn). Tính duy nhất qua mạng được yêu cầu để hỗ trợ các hệ thống tập tin mạng. Nhân duy trì một cấu trúc vnode cho mỗi nút hoạt động (tập tin hay thư mục).

Do đó, VFS có sự khác biệt các tập tin cục bộ với các tập tin ở xa, và các tập tin cục bộ được phân biệt dựa theo loại hệ thống tập tin của chúng.

VFS kích hoạt các thao tác đặc tả hệ thống tập tin để quản lý các yêu cầu cục bộ dựa theo các loại hệ thống tập tin và ngay cả các lời gọi các thủ tục giao thức NFS cho các yêu cầu ở xa. Quản lý tập tin được xây dựng từ vnode tương ứng và được truyền như các tham số tới các thủ tục này. Tầng cài đặt kiểu hệ thống tập tin, hay giao thức hệ thống tập tin ở xa, là tầng dưới cùng của kiến trúc.



Hình X-4 Hình ảnh dạng lưu đồ của hệ thống tập tin ảo

Cài đặt thư mục

Chọn giải thuật cấp phát thư mục và quản lý thư mục có tác động lớn đến tính hiệu quả, năng lực, khả năng tin cậy của hệ thống tập tin. Do đó, chúng ta cần hiểu sự thoả hiệp liên quan trong các giải thuật này.

Danh sách tuyến tính

Phương pháp đơn giản nhất cho việc cài đặt thư mục là dùng một danh sách tuyến tính chứa tên tập tin với con trỏ chỉ tới các khối dữ liệu. một danh sách tuyến tính với các mục từ thư mục yêu cầu tìm kiếm tuyến tính để xác định một mục từ cụ thể. Phương pháp này đơn giản để lập trình nhưng mất nhiều thời gian để thực thi.

- Để tạo tập tin mới, trước tiên chúng ta phải tìm thư mục để đảm bảo rằng không có tập tin nào tồn tại với cùng một tên. Sau đó, chúng ta thêm một mục từ mới vào cuối thư mục.
- Để xoá một tập tin, chúng ta tìm kiếm thư mục cho tập tin được xác định bởi tên, sau đó giải phóng không gian được cấp phát tới nó.
- Để dùng lại mục từ thư mục, chúng ta có thể thực hiện một vài bước. Chúng ta có thể đánh dấu mục từ như không được dùng (bằng cách gán nó một tên đặc biệt, như một tên trống hay với một bit xác định trạng thái được dùng hoặc không được dùng trong mỗi mục từ), hay chúng ta có thể gán nó tới một danh sách của các mục từ thư mục trống. Một thay đổi thứ ba là chép mục từ cuối cùng trong thư mục vào vị trí trống và giảm chiều dài của thư mục. Một danh sách liên kết có thể được dùng để giảm thời gian xoá một tập tin.

Bất lợi thật sự của danh sách tuyến tính chứa các mục từ thư mục là tìm kiếm tuyến tính để tìm một tập tin. Thông tin thư mục được dùng thường xuyên và người dùng nhận thấy việc truy xuất tới tập tin là chậm. Để khắc phục nhược điểm này, nhiều hệ điều hành cài đặt một vùng lưu trữ phần mềm (software cache) để lưu hầu hết những thông tin thư mục được dùng gần nhất. Một chập dữ liệu được lưu trữ sẽ tránh đọc lại liên tục thông tin từ đĩa. Một danh sách được sắp xếp cho phép

tìm kiếm nhị phân và giảm thời gian tìm kiếm trung bình. Tuy nhiên, yêu cầu mà một danh sách phải được sắp xếp có thể phức tạp việc tạo và xoá tập tin vì chúng ta phải di chuyển lượng thông tin liên tục để duy trì một thư mục được xếp thứ tự. Một cấu trúc dữ liệu cây tinh vi hơn như B-tree có thể giúp giải quyết vấn đề. Lợi điểm của danh sách được sắp xếp là liệt kê một thư mục có thứ tự mà không cần một bước sắp xếp riêng.

Bảng băm

Một cấu trúc dữ liệu khác thường được dùng cho một thư mục tập tin là bảng băm (hash table). Trong phương pháp này, một danh sách tuyến tính lưu trữ các mục từ thư mục nhưng một cấu trúc bảng băm cũng được dùng. Bảng băm lấy một giá trị được tính từ tên tập tin và trả về con trỏ chỉ tới tên tập tin trong danh sách tuyến tính. Do đó, nó có thể giảm rất lớn thời gian tìm kiếm thư mục. Chèn và xoá cũng tương đối đơn giản mặc dù có thể phát sinh đụng độ-những trường hợp có hai tên tập tin được băm cùng vị trí. Khó khăn chính với một bảng băm là kích thước của nó thường cố định và phụ thuộc vào hàm băm trên kích thước đó.

Thí dụ, giả sử rằng chúng ta thực hiện một bảng băm thăm dò tuyến tính quản lý 64 mục từ. Hàm băm chuyển các tập tin thành các số nguyên từ 0 tới 63, thí dụ bằng cách dùng số dư của phép chia cho 64. Sau đó, nếu chúng ta cố tạo tập tin thứ 65, chúng ta phải mở rộng bảng băm thư mục-tới 128 mục từ. Kết quả là chúng ta cần hàm băm mới phải ánh xạ tới dãy 0-127 và chúng ta phải sắp xếp lại các mục từ thư mục đã có để phản ánh giá trị hàm băm mới. Một cách khác, một bảng băm vòng có thể được dùng. Mỗi mục từ băm có thể là danh sách liên kết thay vì chỉ một giá trị riêng và chúng ta có thể giải quyết các đụng độ bằng cách thêm mục từ mới vào danh sách liên kết. Tìm kiếm có thể chậm vì tìm kiếm một tên có thể yêu cầu từ bước thông qua một danh sách liên kết của các mục từ bảng đụng độ; nhưng điều này vẫn nhanh hơn tìm kiếm tuyến tính qua toàn thư mục.

Các phương pháp cấp phát

Tính tự nhiên của truy xuất trực tiếp đĩa cho phép chúng ta khả năng linh hoạt trong việc cài đặt tập tin. Trong hầu hết mọi trường hợp, nhiều tập tin sẽ được lưu trên cùng đĩa. Vấn đề chính là không gian cấp phát tới các tập tin này như thế nào để mà không gian đĩa được sử dụng hiệu quả và các tập tin có thể được truy xuất nhanh chóng. Ba phương pháp quan trọng cho việc cấp phát không gian đĩa được sử dụng rộng rãi: cấp phát kề, liên kết và chỉ mục. Mỗi phương pháp có ưu và nhược điểm. Một số hệ thống hỗ trợ cả ba. Thông dụng hơn, một hệ thống sẽ dùng một phương pháp cụ thể cho tất cả tập tin.

Cấp phát kề

Phương pháp cấp phát kề yêu cầu mỗi tập tin chiếm một tập hợp các khối kề nhau trên đĩa. Các địa chỉ đĩa định nghĩa một thứ tự tuyến tính trên đĩa. Với thứ tự này, giả sử rằng chỉ một công việc đang truy xuất đĩa, truy xuất khối b+1 sau khi khối b không yêu cầu di chuyển trước. Khi di chuyển đầu đọc được yêu cầu (từ cung từ cuối cùng của cylinder tới cung từ đầu tiên của cylinder tiếp theo), nó chỉ di chuyển một rãnh (track). Do đó, số lượng tìm kiếm đĩa được yêu cầu cho truy xuất kề tới các tập tin được cấp phát là nhỏ nhất, như là thời gian tìm kiếm khi tìm kiếm cuối cùng được yêu cầu. Hệ điều hành IBM VM/CMS dùng cấp phát kề.

Cấp phát kề của một tập tin được định nghĩa bởi địa chỉ đĩa và chiều dài (tính bằng đơn vị khối) của khối đầu tiên. Nếu tập tin có n khối và bắt đầu tại khối b thì nó chiếm các khối b, b+1, b+2,..,b+n-1. Mục từ thư mục cho mỗi tập tin hiển thị địa chỉ của khối bắt đầu và chiều dài của vùng được cấp phát cho tập tin này (hình X-5).

directory

file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

Hình X-5 Không gian đĩa được cấp phát kề

Truy xuất một tập tin được cấp phát kề rất dễ. Đối với truy xuất tuần tự, hệ thống tập tin nhớ địa chỉ đĩa của khối cuối cùng được tham chiếu và đọc khối tiếp theo. Để truy xuất khối i của tập tin mà bắt đầu tại khối b, chúng ta có thể lập tức truy xuất khối b+i. Do đó, cả truy xuất tuần tự và truy xuất trực tiếp có thể được hỗ trợ bởi cấp phát kề.

Tuy nhiên, cấp phát kề có một số vấn đề. Một khó khăn là tìm không gian cho một tập tin mới. Việc cài đặt của hệ thống quản lý không gian trống xác định tác vụ này được hoàn thành như thế nào. Bất cứ hệ thống quản lý nào có thể được dùng nhưng nhanh chậm khác nhau.

Vấn đề cấp phát không gian kề có thể được xem là vấn đề cấp phát lưu trữ động của ứng dụng để thoả mãn yêu cầu kích thước n từ danh sách các lỗ trống. First fit và best fit là những chiến lược chung nhất được dùng để chọn lỗ trống từ tập hợp các lỗ trống sẵn dùng. Những mô phỏng hiển thị rằng cả hai first fit và best fit hiệu quả hơn worst fit về thời gian và sử dụng không gian lưu trữ. First fit hay best fit đều không phải là giải thuật tốt nhất nhưng thường thì first fit nhanh hơn best fit.

Các giải thuật này gặp phải vấn đề phân mãnh ngoài. Khi các tập tin được cấp phát và xoá, không gian đĩa trống bị chia thành những mảnh

nhỏ. Phân mãnh ngoài tồn tại bất cứ khi nào không gian trống được chia thành những đoạn. Nó trở thành một vấn đề khi đoạn kề lớn nhất không đủ cho một yêu cầu; lưu trữ được phân thành nhiều lỗ, không lỗ nào đủ lớn để lưu dữ liệu. Phụ thuộc vào tổng lượng lưu trữ đĩa và kích thước tập tin trung bình, phân mãnh ngoài có thể là vấn đề chính hay phụ.

Một số hệ thống vi tính cũ dùng cấp phát kề trên đĩa mềm. Để ngăn ngừa lượng lớn không gian đĩa phân mãnh ngoài, người dùng phải chạy thủ tục đóng gói lại. Thủ tục này chép toàn bộ hệ thống tập tin tới một đĩa khác hay trên băng từ. Kế đến, đĩa mềm ban đầu được giải phóng hoàn toàn, tạo một không gian trống kề lớn. Sau đó, thủ tục này chép các tập tin trở lại trên đĩa mềm bằng cách cấp phát không gian kề từ một lỗ lớn. Cơ chế này hiệu quả trong việc hợp nhất (compacts) tất cả không gian trống thành một không gian trống kề, giải quyết vấn đề phân mãnh. Chi phí cho việc hợp nhất là thời gian. Chi phí thời gian phục vụ cho các đĩa cứng lớn dùng cấp phát kề, ở đây hợp nhất tất cả không gian mất hàng giờ. Trong thời gian này, các thao tác hệ thống thông thường không thể được phép vì thế hợp nhất tránh được tất cả chi phí do có thay đổi về dữ liệu.

Một vấn đề khác với cấp phát kề là xác định bao nhiêu không gian được yêu cầu cho một tập tin. Khi một tập tin được tạo, toàn bộ không gian nó cần phải được tìm kiếm và được cấp phát. Người tạo (chương trình hay người) biết kích thước tập tin được tạo như thế nào? Trong một số trường hợp việc xác định này tương đối đơn giản (thí dụ chép một tập tin đã có); tuy nhiên, kích thước của tập tin xuất có thể khó để ước lượng.

Nếu chúng ta cấp quá ít không gian tới một tập tin, chúng ta thấy rằng tập tin không thể mở rộng. Đặc biệt với một chiến lược cấp phát best fit, không gian trên cả hai phía của tập tin đang được dùng. Do đó, chúng ta không thể làm cho tập tin lớn hơn. Hai khả năng có thể. Thứ nhất, chương trình người dùng có thể được kết thúc với một thông báo lỗi hợp lý. Sau đó, người dùng phải cấp phát nhiều không gian hơn và chạy chương trình lại. Việc lặp này có thể gây ra chi phí. Để ngăn chặn chúng,

người dùng sẽ mô phỏng nhiều hơn lượng không gian được yêu cầu. Điều này dẫn đến không gian bị lãng phí.

Một khả năng khác là tìm một lỗ trống lớn hơn, chép nội dung của tập tin tới không gian trống mới, và giải phóng không gian trước đó. Một loạt các hoạt động này có thể được lặp lại với điều kiện là không gian tồn tại mặc dù nó tiêu tốn nhiều thời gian. Tuy nhiên, trong trường hợp này người dùng không bao giờ yêu cầu được thông báo về những gì đang xảy ra; hệ thống tiếp tục mặc dù vấn đề phát sinh.

Ngay cả nếu toàn bộ không gian được yêu cầu cho tập tin được biết trước, cấp phát trước là không đủ. Một tập tin sẽ lớn lên trong khoảng thời gian dài phải được cấp phát đủ không gian cho kích thước cuối cùng của nó mặc dù không gian đó có thể không được dùng cho khoảng thời gian dài. Do đó, tập tin có lượng lớn phân mãnh trong.

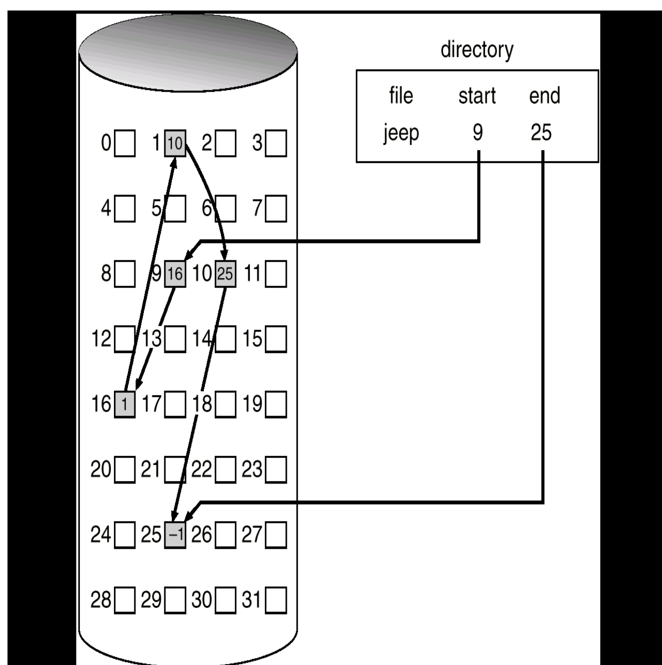
Để tối thiểu các khó khăn này, một số hệ điều hành dùng một cơ chế cấp phát kề được hiệu chỉnh. Trong cơ chế này đoạn không gian kề được cấp phát trước và sau đó khi lượng không gian đó không đủ lớn, một đoạn không gian kề khác, một đoạn mở rộng (extent), được thêm vào cấp phát ban đầu. Sau đó, vị trí của các khối tập tin được ghi lại như một vị trí và một bộ đếm khối cộng với một liên kết tới khối đầu tiên của đoạn mở rộng tiếp theo. Trên một số hệ thống, người sở hữu tập tin có thể đặt kích thước đoạn mở rộng, nhưng việc đặt này có thể không hiệu quả nếu người sở hữu không đúng. Phân mãnh trong vẫn còn là vấn đề nếu đoạn mở rộng quá lớn và phân mãnh ngoài có thể là vấn đề khi các đoạn mở rộng có kích thước khác nhau được cấp phát và thu hồi.

Cấp phát liên kết

Cấp phát liên kết giải quyết vấn đề của cấp phát kề. Với cấp phát liên kết, mỗi tập tin là một danh sách các khối đĩa được liên kết; các khối đĩa có thể được phân tán khắp nơi trên đĩa. Thư mục chứa một con trỏ chỉ tới khối đầu tiên và các khối cuối cùng của tập tin. Thí dụ, một tập tin có 5 khối có thể bắt đầu tại khối số 9, tiếp tục là khối 16, sau đó khối 1,

khối 10 và cuối cùng khối 25 (như hình X-6). Mỗi khối chứa một con trỏ chỉ tới khối kế tiếp. Các con trỏ này không được làm sẵn dùng cho người dùng. Do đó, nếu mỗi khối là 512 bytes, và địa chỉ đĩa (con trỏ) yêu cầu 4 bytes thì phần chứa dữ liệu của khối là 508 bytes.

Để tạo một tập tin mới, chúng ta đơn giản tạo một mục từ mới trong thư mục. Với cấp phát liên kết, mỗi mục từ thư mục có một con trỏ chỉ tới khối đĩa đầu tiên của tập tin. Con trỏ này được khởi tạo tới nil (giá trị con trỏ cuối danh sách) để chỉ một tập tin rỗng. Trường kích thước cũng được đặt tới 0. Một thao tác viết tới tập tin làm một khối trống được tìm thấy bằng hệ thống quản lý không gian trống, sau đó khối mới này được viết tới và được liên kết tới cuối tập tin. Để đọc một tập tin, chúng ta đơn giản đọc các khối bằng cách lần theo các con trỏ từ khối này tới khối khác. Không có sự phân mãnh ngoài với cấp phát liên kết, và bất cứ khối trống trên danh sách không gian trống có thể được dùng để thoả mãn yêu cầu. Kích thước của một tập tin không cần được khai báo khi tập tin đó được tạo. Một tập tin có thể tiếp tục lớn lên với điều kiện là các khối trống sẵn có. Do đó, nó không bao giờ cần thiết để hợp nhất không gian trống.



Hình X-6 cấp phát không gian đĩa liên kết

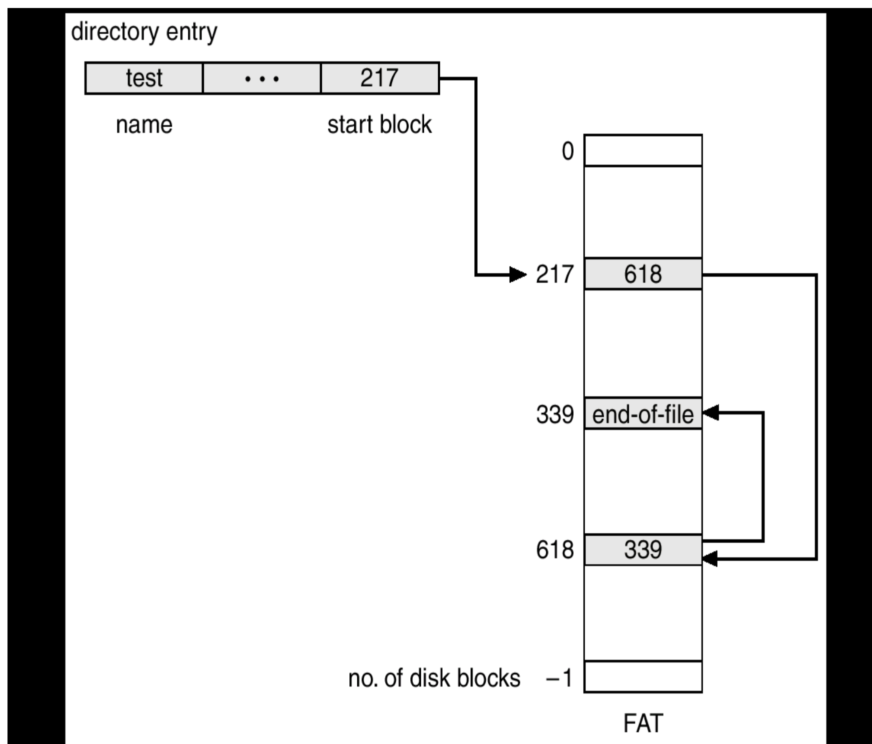
Tuy nhiên, cấp phát liên kết có một vài nhược điểm. Vấn đề chủ yếu là nó có thể được dùng hiệu quả chỉ cho các tập tin truy xuất tuần tự. Để tìm khối thứ i của tập tin, chúng ta phải bắt đầu tại điểm bắt đầu của tập tin đó, và lần theo con trỏ cho đến khi chúng ta nhận được khối thứ i. Mỗi truy xuất tới con trỏ yêu cầu một thao tác đọc đĩa, và đôi khi là một tìm kiếm đĩa. Do đó, nó không đủ hỗ trợ một khả năng truy xuất trực tiếp cho các tập tin cấp phát liên kết.

Một nhược điểm khác của cấp phát liên kết là không gian được yêu cầu cho các con trỏ. Nếu một con trỏ yêu cầu 4 bytes của khối 512 bytes thì 0.77% của đĩa được dùng cho các con trỏ thay vì là thông tin.

Một giải pháp thông thường để giải quyết vấn đề này là tập hợp các khối vào các nhóm (clusters) và cấp phát các nhóm hơn là các khối. Thí dụ, hệ thống tập tin có thể định nghĩa nhóm gồm 4 khối và thao tác trên đĩa chỉ trong đơn vị nhóm thì các con trỏ dùng % nhỏ hơn của không gian của tập tin. Phương pháp này cho phép ánh xạ khối luận lý tới vật lý vẫn còn đơn giản, nhưng cải tiến thông lượng đĩa và giảm không gian được yêu cầu cho cấp phát khối và quản lý danh sách trống. Chi phí của tiếp cận này là tăng phân mãnh trong vì nhiều không gian hơn bị lãng phí nếu một nhóm chỉ đầy một phần hơn là một khối đầy một phần. Các nhóm có thể được dùng để cải tiến thời gian truy xuất đĩa cho nhiều giải thuật khác nhau vì thế chúng được dùng trong hầu hết các hệ điều hành.

Một vấn đề khác của cấp phát liên kết là khả năng tin cậy. Vì các tập tin được liên kết với nhau bởi các con trỏ được phân tán khắp đĩa, xem xét điều gì xảy ra nếu một con trỏ bị mất hay bị phá hỏng. Một con bọ (bug) trong phần mềm hệ điều hành hay lỗi phần cứng đĩa có thể dẫn tới việc chọn con trỏ sai. Lỗi này có thể dẫn tới việc liên kết vào danh sách không gian trống hay vào một tập tin khác. Các giải pháp một phần là dùng các danh sách liên kết đôi hay lưu tên tập tin và số khối tương đối trong mỗi khối; tuy nhiên, các cơ chế này yêu cầu nhiều chi phí hơn cho mỗi tập tin.

Một thay đổi quan trọng trên phương pháp cấp phát liên kết là dùng bảng cấp phát tập tin (file allocation table-FAT). Điều này đơn giản nhưng là phương pháp cấp phát không gian đĩa hiệu quả được dùng bởi hệ điều hành MS-DOS và OS/2. Một phần đĩa tại phần bắt đầu của mỗi phân khu được thiết lập để chứa bảng này. Bảng này có một mục từ cho mỗi khối đĩa và được lập chỉ mục bởi khối đĩa. FAT được dùng nhiều như là một danh sách liên kết. Mục từ thư mục chứa số khối của khối đầu tiên trong tập tin. Mục từ bảng được lập chỉ mục bởi số khối đó sau đó chứa số khối của khối tiếp theo trong tập tin. Chuỗi này tiếp tục cho đến khi khối cuối cùng, có giá trị cuối tập tin đặc biệt như mục từ bảng. Các khối không được dùng được hiển thị bởi giá trị bảng 0. Cấp phát một khối mới tới một tập tin là một vấn đề đơn giản cho việc tìm mục từ bảng có giá trị 0 đầu tiên và thay thế giá trị kết thúc tập tin trước đó với địa chỉ của khối mới. Sau đó, số 0 được thay thế với giá trị kết thúc tập tin. Một thí dụ minh hoạ là cấu trúc FAT của hình X-7 cho một tập tin chứa các khối đĩa 217, 618 và 339.



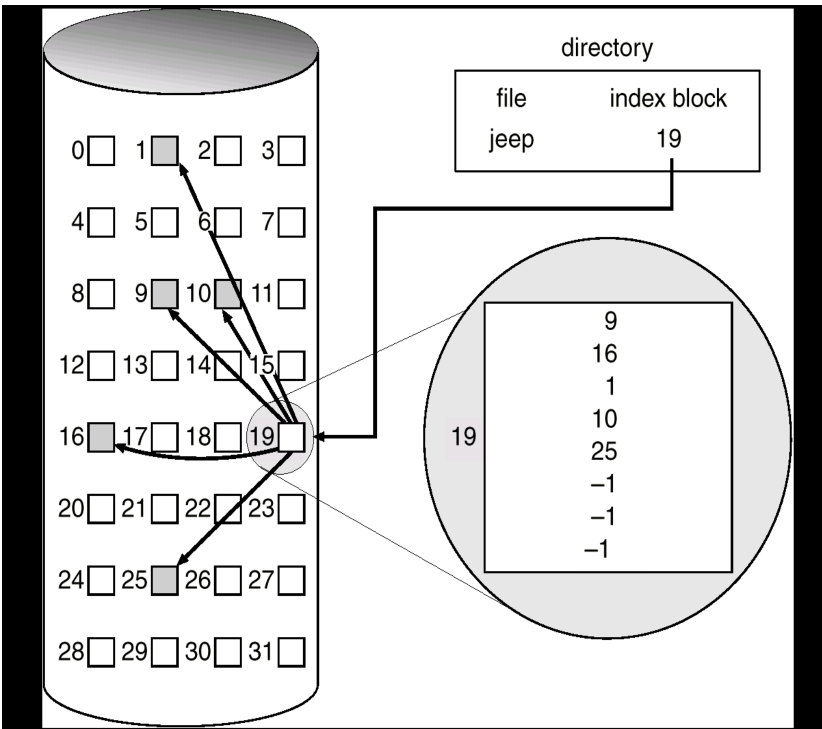
Hình X-7 Bảng cấp phát tập tin

Cơ chế cấp phát FAT có thể dẫn tới số lượng lớn tìm kiếm đầu đọc đĩa nếu FAT không được lưu trữ(cache). Đầu đọc đĩa phải di chuyển tới điểm bắt đầu của phân khu để đọc FAT và tìm vị trí khối sau đó di chuyển tới vị trí của chính khối đĩa đó. Trong trường hợp xấu nhất, cả hai di chuyển xảy ra cho mỗi khối đĩa. Lợi điểm là thời gian truy xuất ngẫu nhiên được cải tiến vì đầu đọc đĩa có thể tìm vị trí của bất cứ khối nào bằng cách đọc thông tin trong FAT.

Cấp phát được lập chỉ mục

Cấp phát liên kết giải quyết việc phân mãnh ngoài và vấn đề khai báo kích thước của cấp phát kề. Tuy nhiên, cấp phát liên kết không hỗ trợ truy xuất trực tiếp hiệu quả vì các con trỏ chỉ tới các khối được phân tán với chính các khối đó qua đĩa và cần được lấy lại trong thứ tự. Cấp phát được lập chỉ mục giải quyết vấn đề này bằng cách mang tất cả con trỏ vào một vị trí: khối chỉ mục (index block).

Mỗi tập tin có khối chỉ mục của chính nó, khối này là một mảng các địa chỉ khối đĩa. Mục từ thứ i trong khối chỉ mục chỉ tới khối i của tập tin. Thư mục chứa địa chỉ của khối chỉ mục (như hình X-8). Để đọc khối i, chúng ta dùng con trỏ trong mục từ khối chỉ mục để tìm và đọc khối mong muốn. Cơ chế này tương tự như cơ chế phân trang.



Hình X-8 Cấp phát không gian đĩa được lập chỉ mục

Khi một tập tin được tạo, tất cả con trỏ trong khối chỉ mục được đặt tới nil. Khi khối thứ i được viết đầu tiên, khối được chứa từ bộ quản lý không gian trống và địa chỉ của nó được đặt trong mục từ khối chỉ mục.

Cấp phát được lập chỉ mục hỗ trợ truy xuất trực tiếp, không gặp phải sự phân mãnh ngoài vì bất cứ khối trống trên đĩa có thể đáp ứng yêu cầu thêm không gian.

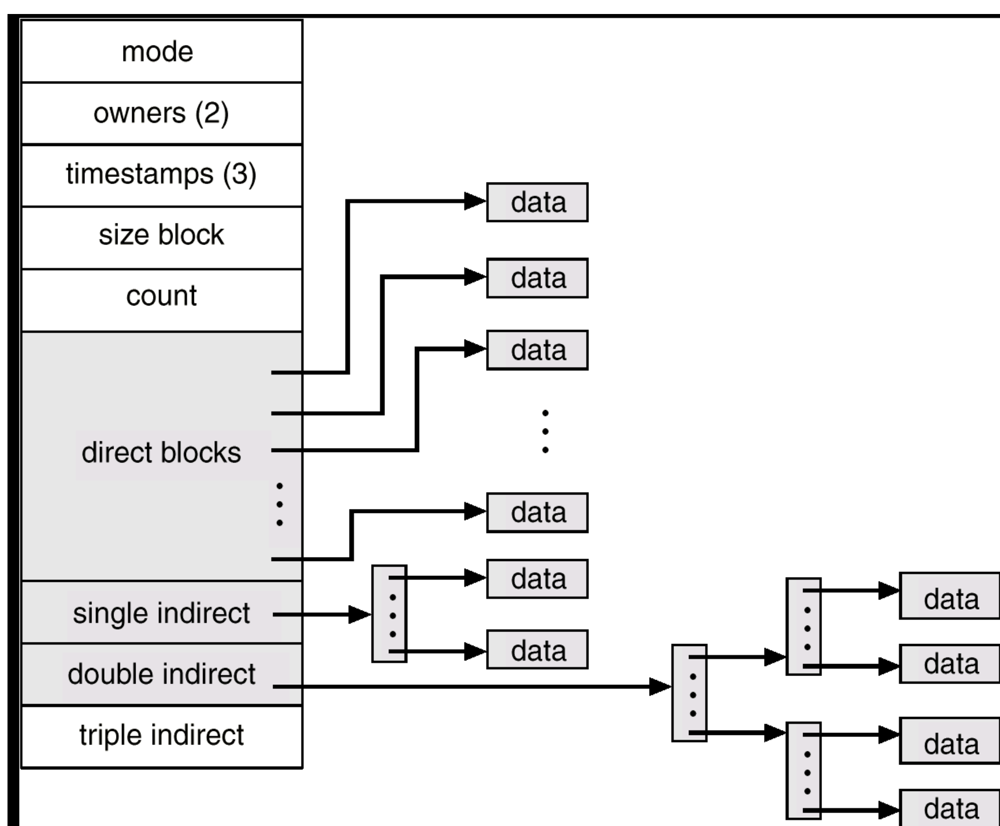
Cấp phát được lập chỉ mục gặp phải sự lãng phí không gian. Chi phí con trỏ của khối chỉ mục thường lớn hơn chi phí con trỏ của cấp phát liên kết. Xét trường hợp thông thường trong đó chúng ta có một tập tin với chỉ một hoặc hai khối. Với cấp phát liên kết, chúng ta mất không gian của chỉ một con trỏ trên khối (một hay hai con trỏ). Với cấp phát được lập chỉ mục, toàn bộ khối chỉ mục phải được cấp phát thậm chí nếu một hay hai con trỏ là khác nil.

Điểm này sinh ra câu hỏi khối chỉ mục nên lớn bao nhiêu? Mỗi tập tin phải có một khối chỉ mục để mà chúng ta muốn khối chỉ mục nhỏ nhất

có thể. Tuy nhiên, nếu khối chỉ mục quá nhỏ nó không thể quản lý đủ các con trỏ cho một tập tin lớn và một cơ chế sẽ phải sẵn có để giải quyết vấn đề này:

- Cơ chế liên kết (linked scheme): một khối chỉ mục thường là một khối đĩa. Do đó, nó có thể được đọc và viết trực tiếp bởi chính nó. Để cho phép đối với các tập tin lớn, chúng ta có thể liên kết nhiều khối chỉ mục với nhau. Thí dụ, một khối chỉ mục có thể chứa một header nhỏ cho tên tập tin và một tập hợp của các địa chỉ 100 khối đĩa đầu tiên. Địa chỉ tiếp theo (từ cuối cùng trong khối chỉ mục) là nil (đối với một tập tin nhỏ) hay một con trỏ tới khối chỉ mục khác (cho một tập tin lớn)
- Chỉ mục nhiều cấp (multilevel index): một biến dạng của biểu diễn liên kết là dùng khối chỉ mục cấp 1 để chỉ tới khối chỉ mục cấp 2. Khối cấp 2 chỉ tới các khối tập tin. Để truy xuất một khối, hệ điều hành dùng chỉ mục cấp 1 để tìm một khối chỉ mục cấp 2 và khối đó tìm khối dữ liệu mong muốn. Tiếp cận này có thể được tiếp tục tới cấp 3 hay cấp 4, tuỳ thuộc vào kích thước tập tin lớn nhất được mong muốn. Với khối có kích thước 4,096 bytes, chúng ta có thể lưu 1,024 con trỏ 4 bytes trong một khối chỉ mục. Chỉ mục hai cấp cho phép 1,048,576 khối dữ liệu, cho phép tập tin có kích thước tới 4GB.
- Cơ chế kết hợp (combined scheme): một biến dạng khác được dùng trong UFS là giữ 15 con trỏ đầu tiên của khối chỉ mục trong inode của tập tin. 12 con trỏ đầu tiên của 15 con trỏ này chỉ tới khối trực tiếp (direct blocks); nghĩa là chúng chứa các địa chỉ của khối mà chứa dữ liệu của tập tin. Do đó, dữ liệu đối với các tập tin nhỏ (không lớn hơn 12 khối) không cần một khối chỉ mục riêng. Nếu kích thước khối là 4 KB, thì tới 48 KB dữ liệu có thể truy xuất trực tiếp. 3 con trỏ tiếp theo chỉ tới các khối gián tiếp (indirect blocks). Con trỏ khối gián tiếp thứ nhất là địa chỉ của khối gián tiếp đơn (single indirect blocks). Khối gián tiếp đơn là một khối chỉ mục không chứa dữ liệu nhưng chứa địa chỉ của các khối chứa dữ liệu. Sau đó, có con trỏ khối gián tiếp đôi (double indirect block) chứa địa chỉ của một khối mà khối này chứa địa chỉ của các khối chứa con trỏ chỉ tới khối dữ liệu thật sự. Con trỏ cuối cùng chứa chứa địa chỉ của khối gián tiếp ba (triple indirect block). Với phương pháp này, số

khối có thể được cấp phát tới một tập tin vượt quá lượng không gian có thể đánh địa chỉ bởi các con trỏ tập tin 4 bytes hay 4 GB. Nhiều cài đặt UNIX gồm Solaris và AIX của IBM hỗ trợ tới 64 bit con trỏ tập tin. Các con trỏ có kích thước này cho phép các tập tin và hệ thống tập tin có kích thước tới terabytes. Một inode được hiển thị trong hình X-9:



Hình X-9 Inode của UNIX

Cơ chế cấp phát lập chỉ mục gặp một số khó khăn về năng lực như cấp phát liên kết. Đặc biệt, các khối chỉ mục có thể được lưu trữ (cache) trong bộ nhớ; nhưng các khối dữ liệu có thể được trãi rộng khắp phân khu.

Năng lực

Các phương pháp cấp phát ở trên khác nhau về tính hiệu quả lưu trữ và thời gian truy xuất khối dữ liệu. Cả hai yếu tố này là tiêu chuẩn quan trọng trong việc chọn phương pháp hợp lý hay các phương pháp cho một hệ điều hành cài đặt.

Trước khi chọn một phương pháp, chúng ta cần xác định hệ thống sẽ được dùng như thế nào. Một hệ thống với hầu hết truy xuất tuần tự nên dùng một phương pháp khác từ hệ thống với hầu hết truy xuất ngẫu nhiên. Đối với bất cứ loại truy xuất nào, cấp phát kề yêu cầu chỉ một truy xuất để đạt được một khối đĩa. Vì chúng ta có thể giữ dễ dàng địa chỉ khởi đầu của tập tin trong bộ nhớ, chúng ta có thể tính lập tức địa chỉ đĩa của khối thứ i (hay khối kế tiếp) và đọc nó trực tiếp.

Đối với cấp phát liên kết, chúng ta cũng có thể giữ địa chỉ khối kế tiếp trong bộ nhớ và đọc nó trực tiếp. Phương pháp này là tốt cho truy xuất tuần tự; tuy nhiên, đối với truy xuất trực tiếp một truy xuất tới khối thứ i phải yêu cầu đọc I đĩa. Vấn đề này minh hoạ lý do cấp phát liên kết không được dùng cho một ứng dụng yêu cầu truy xuất trực tiếp.

Do đó, một số hệ thống hỗ trợ các tập tin truy xuất trực tiếp bằng cách dùng cấp phát kề và truy xuất tuần tự bởi cấp phát liên kết. Đối với các hệ thống này, loại truy xuất được thực hiện phải được khai báo khi tập tin được tạo. Một tập tin được tạo cho truy xuất tuần tự sẽ được liên kết và không thể được dùng cho truy xuất trực tiếp. Một tập tin được tạo cho truy xuất trực tiếp sẽ kề nhau và có thể hỗ trợ cả hai truy xuất trực tiếp và truy xuất tuần tự nhưng chiều dài tối đa của nó phải được khai báo khi nó được tạo. Trong trường hợp này, hệ điều hành phải có cấu trúc dữ liệu hợp lý và các giải thuật để hỗ trợ cả hai phương pháp cấp phát. Các tập tin có thể được chuyển từ một kiểu này sang một kiểu khác bằng cách tạo một tập tin mới của loại mong muốn và các nội dung của tập tin cũ được chép vào tập tin mới. Sau đó, tập tin cũ có thể bị xoá và tập tin mới được đổi tên.

Cấp phát dạng chỉ mục phức tạp hơn. Nếu khối chỉ mục đã ở trong bộ nhớ rồi thì truy xuất có thể được thực hiện trực tiếp. Tuy nhiên, giữ khối chỉ mục trong bộ nhớ yêu cầu không gian có thể xem xét. Nếu không gian

bộ nhớ này không sẵn dùng thì chúng ta phải đọc trước khối chỉ mục và sau đó khối dữ liệu mong muốn. Đối với chỉ mục hai cấp, đọc hai khối chỉ mục là cần thiết. Đối với tập tin rất lớn, truy xuất một khối gần cuối tập tin yêu cầu đọc tất cả khối chỉ mục để lần theo chuỗi con trỏ trước khi khối dữ liệu được yêu cầu cuối cùng được đọc. Do đó, năng lực của cấp phát chỉ mục phụ thuộc cấu trúc chỉ mục trên kích thước tập tin và vị trí của khối mong muốn.

Một số hệ thống kết hợp cấp phát kề và cấp phát chỉ mục bằng cách dùng cấp phát kề cho các tập tin nhỏ (ba hay bốn khối) và tự động chuyển tới cấp phát chỉ mục nếu tập tin lớn lên. Vì hầu hết các tập tin là nhỏ và cấp phát kề là hiệu quả cho các tập tin nhỏ, năng lực trung bình là rất tốt.

Nhiều tối ưu khác là có thể và đang được dùng. Với sự chênh lệch tốc độ giữa CPU và đĩa, nó là không hợp lý để thêm hàng ngàn chỉ thị tới hệ điều hành để tiết kiệm chỉ một vài di chuyển của đầu đọc. Ngoài ra, sự chênh lệch này tăng theo thời gian, tới điểm nơi mà hàng trăm của hàng ngàn chỉ thị phù hợp có thể được dùng để tối ưu sự di chuyển của đầu đọc.

Quản lý không gian trống

Vì không gian trống là giới hạn nên chúng ta cần dùng lại không gian từ các tập tin bị xoá cho các tập tin mới nếu có thể. Để giữ vết của không gian đĩa trống, hệ thống duy trì một danh sách không gian trống. Danh sách không gian trống ghi lại tất cả khối đĩa trống. Để tạo tập tin, chúng ta tìm trong danh sách không gian trống lượng không gian được yêu cầu và cấp phát không gian đó tới tập tin mới. Sau đó, không gian này được xoá từ danh sách không gian trống. Khi một tập tin bị xoá, không gian đĩa của nó được thêm vào danh sách không gian trống. Mặc dù tên của nó là danh sách nhưng danh sách không gian trống có thể không được cài như một danh sách.

Bit vector

Thường thì danh sách không gian trống được cài đặt như một bản đồ bit (bit map) hay một vector bit (bit vector). Mỗi khối được biểu diễn bởi 1 bit. Nếu khối là trống, bit của nó được đặt là 1, nếu khối được cấp phát bit của nó được đặt là 0.

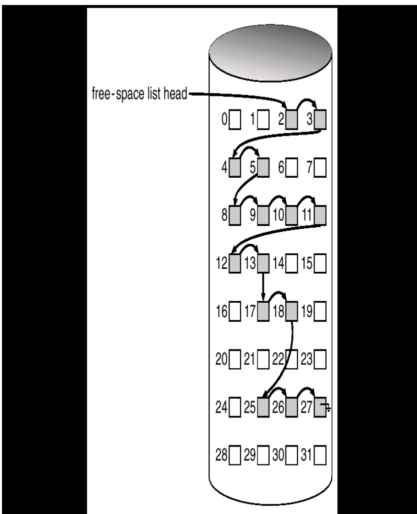
Thí dụ, xét một đĩa khi các khối 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, 18, 25, 26, và 27 là trống và các khối còn lại được cấp phát. Bản đồ bit không gian trống sẽ là:

001111001111110001100000011100000…

Lợi điểm chính của tiếp cận này là tính tương đối đơn giản và hiệu quả của nó trong việc tìm khối trống đầu tiên, hay n khối trống tiếp theo trên đĩa.

Một lần nữa, chúng ta thấy các đặc điểm phần cứng định hướng chức năng phần mềm. Tuy nhiên, các vector bit là không đủ trừ khi toàn bộ vector được giữ trong bộ nhớ chính. Giữ nó trong bộ nhớ chính là có thể cho các đĩa nhỏ hơn, như trên các máy vi tính nhưng không thể cho các máy lớn hơn. Một đĩa 1.3 GB với khối 512 bytes sẽ cần một bản đồ bit 332 KB để ghi lại các khối trống. Gom bốn khối vào một nhóm có thể giảm số này xuống còn 83 KB trên đĩa.

Danh sách liên kết



Hình X-10 danh sách không gian trống được liên kết trên đĩa

Một tiếp cận khác để quản lý bộ nhớ trống là liên kết tất cả khối trống, giữ một con trỏ tới khối trống đầu tiên trong một vị trí đặc biệt trên đĩa và lưu nó trong bộ nhớ. Khối đầu tiên này chứa con trỏ chỉ tới khối đĩa trống tiếp theo,..Trong thí dụ trên, chúng ta có thể giữ một con trỏ chỉ tới khối 2 như là khối trống đầu tiên. Khối 2 sẽ chứa một con trỏ chỉ tới khối 3, khối này sẽ chỉ tới khối 4,…(như hình X-10). Tuy nhiên, cơ chế này không hiệu quả để duyệt danh sách, chúng ta phải đọc mỗi khối, yêu cầu thời gian nhập/xuất đáng kể. Tuy nhiên, duyệt danh sách trống không là hoạt động thường xuyên. Thường thì, hệ điều hành cần một khối trống để mà nó có thể cấp phát khối đó tới một tập tin, vì thế khối đầu tiên trong danh sách trống được dùng. Phương pháp FAT kết hợp với đếm khối trống thành cấu trúc dữ liệu cấp phát.

Nhóm

Thay đổi tiếp cận danh sách trống để lưu địa chỉ của n khối trống trong khối trống đầu tiên. n-1 khối đầu tiên này thật sự là khối trống. Khối cuối cùng chứa địa chỉ của n khối trống khác, …Sự quan trọng của việc cài đặt này là địa chỉ của một số lượng lớn khối trống có thể được tìm thấy nhanh chóng, không giống như trong tiếp cận danh sách liên kết chuẩn.

Bộ đếm

Một tiếp cận khác đạt được lợi điểm trong thực tế là nhiều khối kề có thể được cấp phát và giải phóng cùng lúc, đặc biệt khi không gian được cấp phát với giải thuật cấp phát kề hay thông qua nhóm. Do đó, thay vì giữ một danh sách n địa chỉ đĩa trống, chúng ta có thể giữ địa chỉ của khối trống đầu tiên và số n khối kề trống theo sau khối đầu tiên. Mỗi mục từ trong danh sách không gian trống sau đó chứa một địa chỉ đĩa và bộ đếm. Mặc dù mỗi mục từ yêu cầu nhiều không gian hơn một địa chỉ đĩa đơn, nhưng toàn bộ danh sách sẽ ngắn hơn với điều kiện là bộ đếm lớn hơn 1.

Tóm tắt

Hệ thống tập tin định vị không đổi trên thiết bị lưu trữ phụ được thiết kế để quản lý một lượng lớn dữ liệu không đổi. Phương tiện lưu trữ phụ phổ biến nhất là đĩa.

Đĩa vật lý có thể được chia thành nhiều phân khu để điều khiển việc sử dụng phương tiện và cho phép nhiều hệ thống tập tin (có thể khác nhau) trên đĩa. Các hệ thống tập tin này được gắn vào kiến trúc hệ thống tập tin luận lý để làm cho chúng sẵn dùng. Các hệ thống tập tin thường được cài đặt trong một kiến trúc phân tầng hay module. Những cấp thấp hơn giải quyết các thuộc tính vật lý của các thiết bị lưu trữ. Cấp cao hơn giải quyết các tên tập tin biểu tượng và các thuộc tính luận lý của tập tin. Các cấp trung gian ánh xạ các khái niệm tập tin luận lý thành các thuộc tính thiết bị vật lý.

Mỗi kiểu hệ thống tập tin có các cấu trúc và giải thuật khác nhau. Một tầng VFS cho phép các tầng cao hơn giải quyết mỗi kiểu hệ thống tập tin khác nhau trong cùng một cách. Ngay cả các hệ thống tập tin ở xa có thể được tích hợp vào cấu trúc thư mục của hệ thống và được hoạt động trên các lời gọi hệ thống chuẩn bằng giao diện VFS.

Những tập tin khác nhau có thể được cấp phát không gian trên đĩa trong 3 cách: kề, liên kết hay chỉ mục. Cấp phát kề có thể gặp phải sự phân

mãnh ngoài. Truy xuất trực tiếp là kém hiệu quả với cấp phát liên kết. Cấp phát chỉ mục yêu cầu chi phí đáng kể cho khối chỉ mục của nó. Các giải thuật này có thể tối ưu trong nhiều cách. Không gian kề có thể lớn lên thông qua đoạn mở rộng để tăng khả năng linh hoạt và giảm phân mãnh ngoài. Cấp phát chỉ mục có thể được thực hiện trong việc nhóm nhiều khối để tăng thông lượng và giảm số lượng các mục từ chỉ mục được yêu cầu. Lập chỉ mục trong các nhóm là tương tự như cấp phát kề với các đoạn mở rộng.

Các phương pháp cấp phát không gian trống cũng ảnh hưởng tới tính hiệu quả của không gian đĩa, năng lực hệ thống tập tin và khả năng tin cậy của thiết bị lưu trữ phụ. Các phương pháp được dùng gồm các vector bit và các danh sách liên kết. Các tối ưu gồm nhóm, đếm và FAT, mà đặt danh sách liên kết trong một vùng kề.